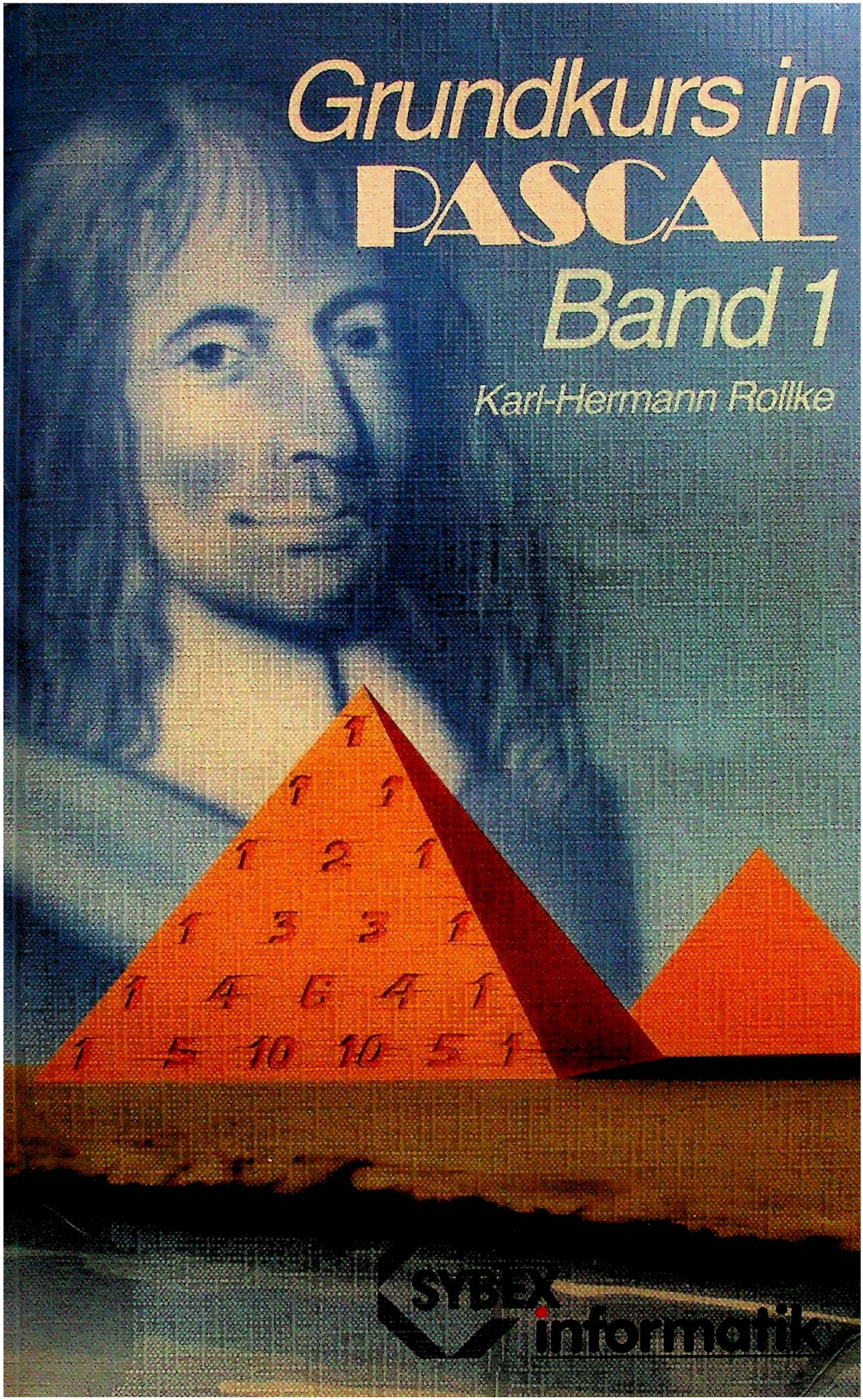
Grundkurs in PASCAL Band 1

Karl-Hermann Rollke



SYBEX informatik

Grundkurs in PASCAL Band 1

Grundkurs in PASCAL Band 1

Karl-Hermann Rollke

SYBEX informatik

Anmerkungen:

Apple II ist ein eingetragenes Warenzeichen von Apple Computer, Inc.
UCSD Pascal ist ein eingetragenes Warenzeichen des Dekanats der University of California, San Diego.

Fast alle Software- und Hardwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden.

Der Verlag hat alle Sorgfalt walten lassen, um vollständige und akkurate Informationen in diesem Buch bzw. Programm und anderen evtl. beiliegenden Informationsträgern zu publizieren. SYBEX Verlag GmbH, Düsseldorf, übernimmt keine Garantie noch die juristische Verantwortung oder irgendeine Haftung für die Nutzung dieser Informationen, für deren Wirtschaftlichkeit oder fehlerfreie Funktion für einen bestimmten Zweck. Ferner kann der Verlag für Schäden, die auf eine Fehlfunktion von Programmen, Schaltplänen o.ä. zurückzuführen sind, nicht haftbar gemacht werden, auch nicht für die Verletzung von Patent- und anderen Rechten Dritter, die daraus resultieren.

Umschlagentwurf: Daniel Boucherie
Satz und Layout: tgr – typo-grafik-repro gmbh, Remscheid
Gesamtherstellung: Druckerei Hub. Hoch, Düsseldorf

ISBN 3-88745-046-9
1.–2. Auflage 1984
3.–6. Auflage 1985
7. Auflage 1986
8. Auflage 1986
9. Auflage 1987
10. Auflage 1987
11. Auflage 1988

Alle Rechte vorbehalten. Die Vervielfältigung und Übertragung, auch einzelner Textabschnitte, Bilder oder Zeichnungen, ist – mit Ausnahme der Vervielfältigungen zum persönlichen oder eigenen Gebrauch gem. §§ 53, 54 URG – ohne schriftliche Zustimmung des Verlages nicht zulässig. Das gilt sowohl für die Vervielfältigung, Fotokopien oder irgendein anderes Verfahren als auch für die Übertragung auf Filme, Bänder, Platten, Arbeitstransparente oder andere Medien.

Printed in Germany
Copyright © 1984 by SYBEX-Verlag GmbH, Düsseldorf

Inhalt

0. Vorwort **11**

1. Einführung **13**

1.1 Zur Systematik des Buches 14
1.2 Grundlagen 15
1.3 Problemanalyse – Planung von Programmen 17

2. Erste Schritte mit dem UCSD-Pascal-Betriebssystem . **21**

2.1 Wir fangen an 22
2.2 Unser erstes Programm 26
2.3 Wir schreiben einen Brief mit dem Editor 28
2.4 Zusammenfassung 30

3. Konzept und Aufbau eines Pascal-Programms **31**

3.1 Das Grundprinzip eines Pascal-Programms und seiner Teile 32
3.2 Ein- und Ausgabebefehle – ein kleines Programm . . . 35
3.3 Konstanten und Variablen 39
3.4 Wie hilft sich der stolze Besitzer eines Druckers? 43
3.5 Pascal kann auch rechnen – Beispiele 44
3.6 Mit Grafik geht manches besser 51
3.7 Zusammenfassung 53

4. Reservierte Wörter und Symbole **55**

4.1 Zusammenstellung der Wörter und Symbole 55
4.2 Syntaxdiagramme, und wie man sie liest 58

5. Einfache Datentypen und deren Ein-/Ausgabeformate . **63**

5.1 Ganze Zahlen – INTEGER 64
5.2 Dezimalzahlen – REAL 69
5.3 Zeichen – CHAR 74
5.4 Zeichenketten (Textverarbeitung) – STRING in UCSD-Pascal 77
5.5 Wahrheitswerte – BOOLEAN 81
5.6 Zusammenfassung 85

6. Immer diese Wiederholungen 87

6.1 Zähle von ... bis ... – die FOR-Schleife 88
6.2 Wiederhole ... bis ... – die REPEAT-Schleife 93
6.3 Solange ... tue ... – die WHILE-Schleife 98
6.4 Zusammenfassung 102

7. Entscheidungen 103

7.1 Wenn ... dann ... sonst ... – die IF-THEN-ELSE Konstruktion 104
7.2 CASE für Mehrfachentscheidungen und Schreibfaule . 109
7.3 Zusammenfassung 113

8. Kleiner Ausflug in die Grafik 115

8.1 Was hat die Schildkröte mit dem Computer zu tun? . . 116
8.2 Befehle der UNIT TURTLEGRAPHICS 117
8.3 Kleine Programme mit Turtles 121
8.4 Andere UNITs 125
8.5 Zusammenfassung 128

9. Besser geht's mit Prozeduren 129

9.1 Wenn wir das Dreieck immer wieder zeichnen wollen. Prozeduren und Prozedurkonzept 130
9.2 Prozeduren können ihre eigenen Variablen haben . . . 135
9.3 Übergabe von Variablen an Prozeduren 140
9.4 Die Prozeduren machen sich an die Variablen des Hauptprogramms heran 144
9.5 Zusammenfassung 148

10. Funktionen 149

10.1 Unterschied Funktion – Prozedur 149
10.2 Übungen zu Funktionen 154
10.3 Zusammenfassung 157

11. Rekursionen 159

11.1 Was ist eine Rekursion? 160
11.2 Probleme der Rechenzeit und des Speicherplatzes mit Rekursionen 164
11.3 Beispiele zu Rekursionen 168
11.4 Zusammenfassung 174

12. Eindimensionale Felder 175

12.1 Wenn mehrere Variablen den gleichen Namen haben sollen 176
12.2 Beispiele eindimensionaler ARRAYs 178
12.3 Zusammenfassung 182

13. Wer Ordnung hält, ist nur zu faul zum Suchen – Sortieralgorithmen 183

13.1 Sortieren durch Austausch (1) 184
13.2 Sortieren durch Austausch (2) – Bubblesort 188
13.3 Quicksort (rekursiv) 191
13.4 Zusammenfassung 193

Anhang:

A: Systemstart / Formatieren / Kopieren / Besondere Zeichen 196
B: Beschreibung der COMMAND-, Editor- und Filer-Befehle 198
C: Reservierte Wörter / Vordefinierte Bezeichner 203
D: Tabelle der wichtigsten Standardprozeduren und -funktionen 204
E: Der ASCII-Zeichensatz 206
F: Compiler-Fehlermeldungen 207
G: Symbole für Fluß- und Blockdiagramme 211
H: Syntaxdiagramme 212
I: Literaturverzeichnis 218
J: Stichwortverzeichnis 219

Programme

Programmbeispiel	Name	Seite
1. Einführendes Programm	TEST	19
2. Erstes Programm	ERSTES	26
3. Struktur eines Programms	VIELECKE	32
4. Ausgabeprogramm	SAETZE	37
5. Ausgabeprogramm	BILDSCHIRMTEST	38
6. Kreisfläche	BEISPIEL	39
7. Testprogramm	VARTEST	42
8. Druckerprogramm	DRUCKTEST	43
9. Rechenprogramm	PAKET	48
10. Rechenprogramm	WINKELTABELLE	49
11. Grafikprogramm	RECHTECK	52
12. Teilen mit Rest	TEILE	66
13. Lange ganze Zahlen	SCHACHKORN	68
14. Kugelberechnung	KUGEL	71
15. Zeichensatz	TESTZEICHEN	75
16. Zeichenketten	WORTTEST	78
17. Zeichenketten	STRINGDEMO	80
18. Wahrheitstabelle	WAHR	83
19. Zählschleife	ASCII	89
20. Zählschleife	DREIECK	91
21. REPEAT-Schleife	WIEDERHOLUNG	95
22. Pi-Berechnung	KREISZAHL	96
23. WHILE-Schleife	QUERSUMME	100
24. Repeat-Schleife	QUERSUMME	101
25. IF...THEN	WASWOHL	105
26. IF...THEN	ZAHLENRATEN	107
27. CASE	BRUCHRECHNUNG	109
28. Grafikprogramm	WHILEPLOT	118
29. Grafikprogramm	DREIECK	122
30. Grafikprogramm	VIELECK	123
31. Grafikprogramm	FENSTERBILD	124
32. Tonprogramm	TONLEITER	127
33. Prozeduren	DREIECK1	130
34. Prozeduren	DREIECK2	132
35. Prozeduren	NA_SO_WAS	133
36. Prozeduren	SCHACHTEL	134
37. Prozeduren	DATUM	138
38. Prozeduren	CALL_BY_VALUE	141
39. Prozeduren	WARENPREIS	142
40. Prozeduren	UNTERSCHIED	144
41. Prozeduren	WENIGERLEER	146
42. Funktionen	POTENZ	152
43. Funktionen	FUNTEST	152

Programmbeispiel	Name	Seite
44. Funktionen	MIN	154
45. Funktionen	GGt	154
46. Funktionen	PRIM	155
47. Funktionen	WEITER	155
48. Funktionen	ZAHLENZAUBER	155
49. Rekursionen	WAS_MACHT_DAS	160
50. Rekursive Grafik	SPIRALE	162
51. Fakultät rekursiv	FAK	163
52. Fakultät iterativ	FAK	163
52. Fibonacci-Zahlen rekursiv	REKUFIB	165
53. Fibonacci-Zahlen iterativ	ITERFIB	165
54. Türme von Hanoi	HANOI	171
55. Türme von Hanoi für Drucker	HANOI	171
56. Rekursive Grafik	SCHNEEFLOCKE	172
57. Array	LISTE	177
58. Array	RANDOMTEST	179
59. Array	LOTTOBLOCK	180
60. Array	MESSWERTE	180
61. Sortieren1	TAUSCHSORT1	186
62. Sortieren1 für Drucker	TAUSCHSORT1	187
63. Sortieren 2	TAUSCHSORT2	189
64. Sortieren2 für Drucker	TAUSCHSORT2	189
65. Sortieren3	QUICKSORT	191
66. Sortieren3 für Drucker	QUICKSORT	192

Vorwort

Kapitel 0

Das vorliegende Buch umfaßt zwei Bände.Themenschwerpunkte der beiden Bände sind:

Themen-Schwerpunkte

Band 1: Einführung in das Programmieren mit Pascal.
Einfache Datenstrukturen.
Algorithmik: Kontrollstrukturen (Entscheidungen und Schleifen), Prozedurkonzept.
Eindimensionale Felder.
Sortieralgorithmen.

Band 2: Datenstrukturen: Verbunde, Mengen, Aufzählungen, Dateien, Zeiger.
Bibliothekskonzept in UCSD-Pascal.
Struktur und Arbeitsweise einer Datenverarbeitungsanlage.
Einblick in die Programmierung auf Maschinenebene und in Assembler.
Gesellschaftliche Probleme der Datenverarbeitung (Datenschutz).
Vorschläge für Projekte.

Die Themen der beiden Bände orientieren sich weitgehend an den (zum Zeitpunkt der Erstellung gültigen) Richtlinien für den Informatik-Unterricht. Da diese Richtlinien von Bundesland zu Bundesland verschieden sind, wurde versucht, ein Konzept zu erstellen, das es erlaubt, durch variable Reihenfolge der Themen das Buch mit unterschiedlichen Richtlinien zu nutzen.

Richtlinien

Außerdem erlaubt die Struktur des Buches auch einen Einsatz im Selbststudium und besonders im Bereich der Weiterbildung und der Volkshochschulen.

Selbststudium Volkshochschule

Die Systematik des Pascal-Teils der beiden Bände ist stark an die Systematik der Programmiersprache Pascal angelehnt, so daß der Leser einen sinnvollen Aufbau erkennen und bestimmte Themen leicht wiederfinden kann.

Systematik

Im Buch werden die wichtigsten Lernelemente ausführlich behandelt, und es wird auf Regeln, Beispiele und Fehlermöglichkeiten hingewiesen. Der Anhang ergänzt mit Zusammenfassungen von Befehlen und Tabellen.

Der verwandte Pascal-Dialekt ist UCSD-Pascal, entwickelt an der University of California in San Diego.

UCSD-Pascal

APPLE II Die Programme und Operationen im UCSD-Betriebssystem sind alle auf einem APPLE II-Rechner entwickelt und getestet worden.

andere Rechner Bei Benutzung anderer Rechner oder anderer Pascal-Versionen wird man möglicherweise Unterschiede in der Sprache und in Funktionen des Rechners feststellen. In diesen Fällen muß das entsprechende Handbuch (und/oder der Lehrer) zur Klärung beitragen.

Das 2. Kapitel bezieht sich direkt auf das UCSD-Betriebssystem und kann bei Benutzung eines anderen Betriebssystems übergangen werden.

Bis auf einige Grafikprogramme (die selbstverständlich nur auf grafikfähigen Rechnern sinnvoll laufen) sollten alle Programme evtl. mit geringfügigen Änderungen) auf allen anderen Rechnern lauffähig sein.

Reihenfolge der Themen Bei der Arbeit mit dem Buch kann der fortgeschrittene Benutzer sicher einige einführende Teile weglassen. Auch die Reihenfolge der Kapitel kann im großen und ganzen verändert werden, wenn ein didaktisch anderes Vorgehen für wünschenswert gehalten wird.

Die meisten Kapitel sind ohne das Lesen anderer Kapitel verständlich. Hin und wieder schaffen kleine Wiederholungen die nötigen Voraussetzungen.

Programmierpraxis Das Buch sollte möglichst in Verbindung mit einem Rechner durchgearbeitet werden. So wie man nicht theoretisch nähen lernen kann, führt auch das Erlernen einer Programmiersprache ohne genügende Praxis und Übung nicht zu einem erfolgreichen Abschluß.

Übungen Von den Übungen zum Schluß eines Paragraphen sollte reichlich Gebrauch gemacht werden.

Dank Das Buch entstand auf der Basis mehrerer Unterrichtsreihen im Grundkurs Informatik, die ich am Geschwister-Scholl-Gymnasium in Unna durchgeführt habe.

Anregungen und konstruktive Kritik verdanke ich den Fachkollegen Bischoff, Geist, Goldkuhle, Greinke, Hagenguth, Humberg, Langewiesche und Reske.

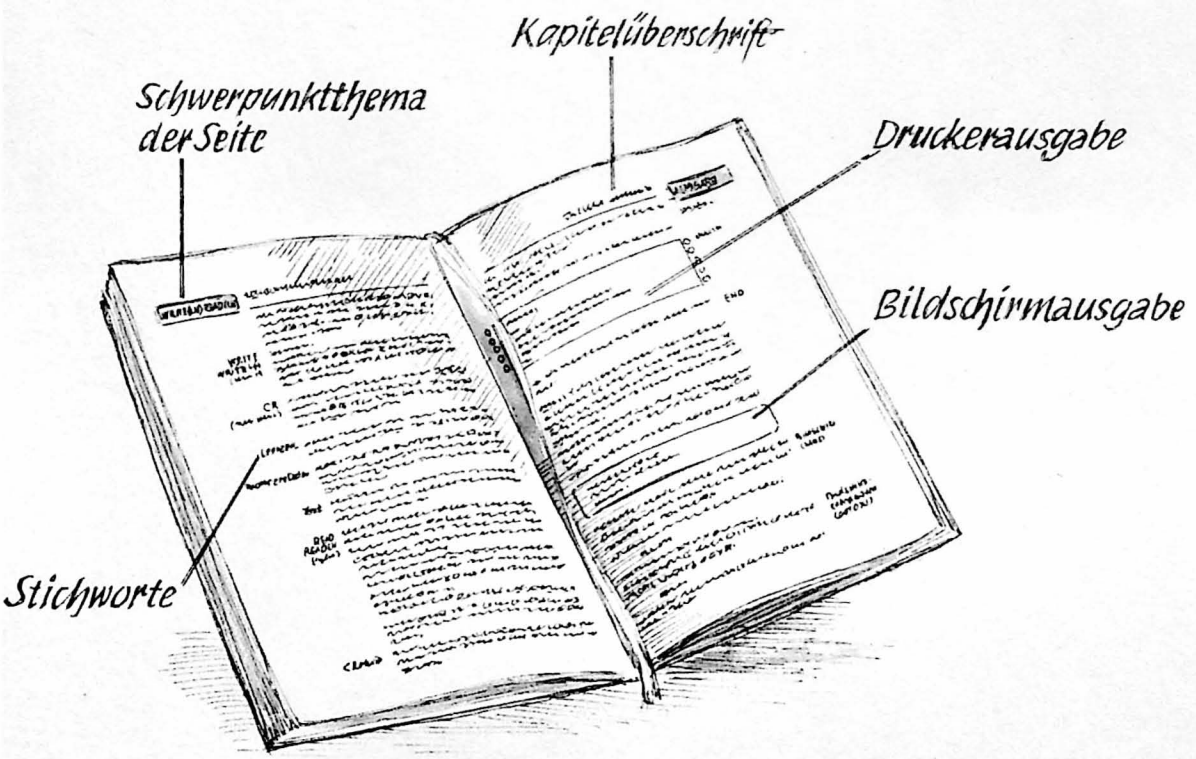
Herrn Dederichs danke ich besonders für die Betreuung des Buches.

Unna, November 1983 Karl-Hermann Rollke

Einführung

Kapitel 1

In diesem Kapitel wird der Aufbau dieses Buches dargestellt. Außerdem werden einige einfache Grundlagen zum Aufbau eines Rechners und zu höheren Programmiersprachen behandelt. Die Planung von Programmen sowie Fehlersuche und Dokumentation von Programmen behandelt ein weiterer Paragraph.



1.1 Zur Systematik des Buches

Kapitel-Aufbau Die Themenschwerpunkte dieses Buches werden Kapiteln zugeordnet.

Aufbau eines Kapitels:

1. Einführung und Lernziele, die nach dem Durcharbeiten des Kapitels erreicht sein sollen.
2. Erarbeitung des Unterrichtsstoffes in Paragraphen.
3. Kurze Zusammenfassung des Erreichten.

Aufbau der Paragraphen Die Paragraphen haben im allgemeinen folgenden Aufbau (soweit die Inhalte es zulassen):

1. Einführende Problemstellung und/oder Beispiel.
2. Lösung des Problems anhand neuer Erkenntnisse.
3. Allgemeine und spezielle Hinweise zur neuen Thematik.
4. Merkkästchen als Zusammenfassung des neu gelernten Stoffes.
5. Übungsaufgaben zur Vertiefung des Stoffes.

Anhang Zum Schluß des Buches findet sich ein Anhang, der die Funktion eines Nachschlagewerks übernehmen soll.

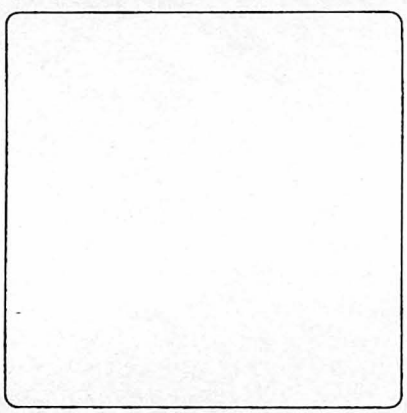
Seiten-Aufbau Zum besseren Auffinden bestimmter Themen in den Kapiteln findet sich am oberen Rand einer Seite:

1. Die Kapitelüberschrift.
2. Das Schwerpunktthema der betreffenden Seite.

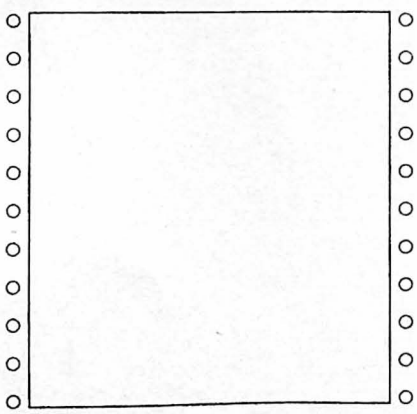
Rand Außerdem wird der Rand jeder Seite zur besseren Übersicht für Stichworte genutzt, die auf das im Text behandelte Thema hinweisen.

Ausgabe Bildschirmausgaben der vorgestellten Programme werden in abgerundeten Kästchen, Programmtexte auf angedeutetem Druckerpapier wiedergegeben:

Bildschirm:

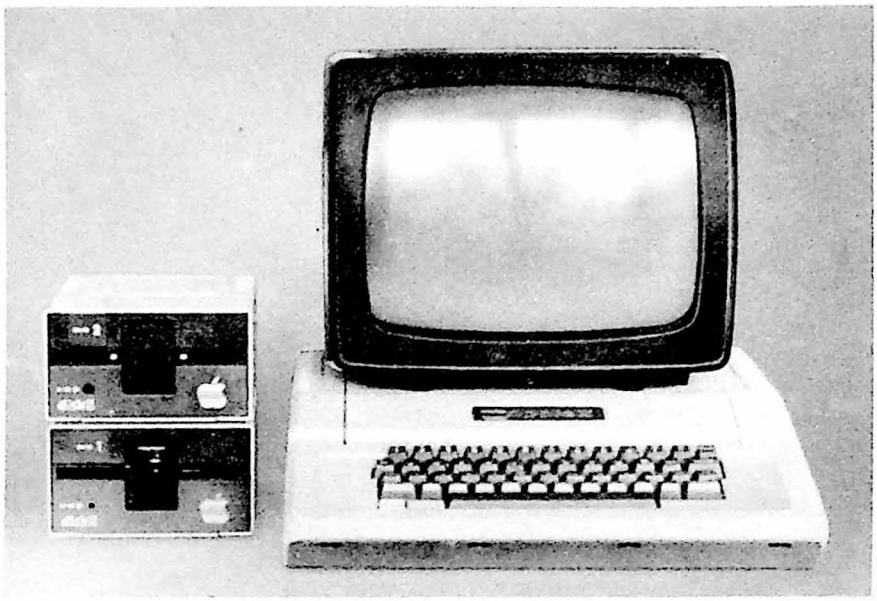


Drucker:



Aufbau eines Computers

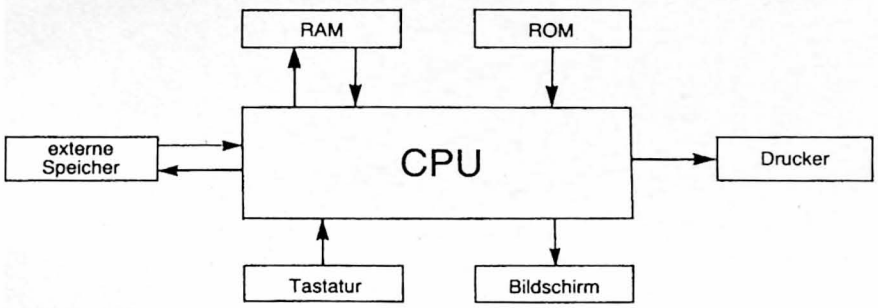
1.2 Grundlagen



Apple II mit Monitor und zwei Diskettenlaufwerken

Grobaufbau eines Computers

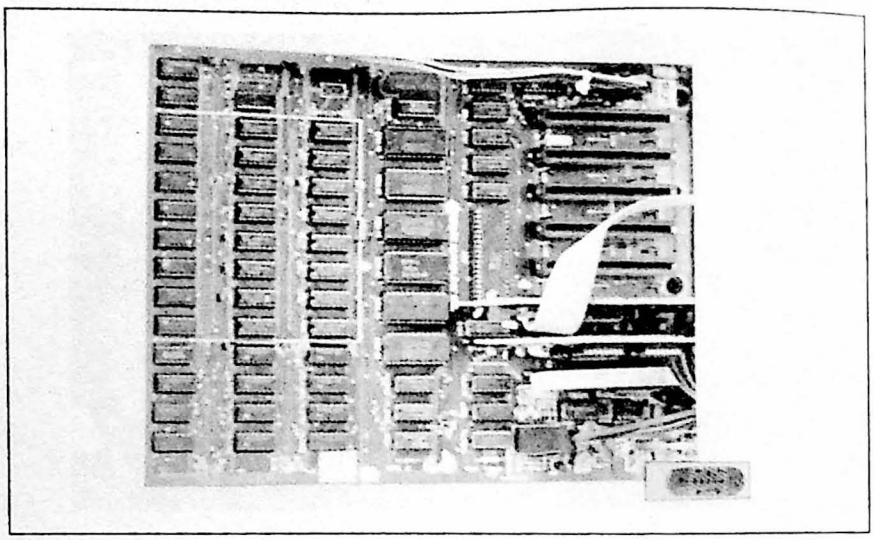
Durch folgenden Grobplan können wir die Funktionseinheiten eines Computers darstellen:



CPU: Central-Processing-Unit, Zentraleinheit.
Hier werden Daten transportiert und verknüpft.

ROM: Read-Only-Memory, Nur-Lese-Speicher.
In diesem Speicher sind Programme abgelegt, die dafür sorgen, daß der Rechner vernünftig funktioniert (z. B., daß nach dem Einschalten des Stroms die Diskettenlaufwerke in Betrieb genommen werden). Beim Ausschalten gehen die Daten nicht verloren.

RAM: Random-Access-Memory, Schreib-Lese-Speicher.
Hier werden Daten und Programme gespeichert, die vom Benutzer eingegeben werden. Nach dem Ausschalten sind die Daten verloren.



Blick in das Innere des Gehäuses

Tastatur: Eingabemedium.

Bildschirm: Ausgabemedium.

Externe Speicher: Bei diesen Speichern handelt es sich um Massenspeicher zur Langzeitspeicherung von Daten und Programmen.

Die wichtigsten Speichermedien sind:

Magnetband Magnetband: Zur sequentiellen Speicherung. Die Daten werden alle hintereinander gespeichert. Will man auf Daten am Ende des Bandes zugreifen, so muß das ganze Band vorgespult werden.

Magnetplatte Magnetplatte: Kreisrunde, magnetisch beschichtete Platte, auf der Daten in konzentrischen Kreisen (Spuren, Tracks) gespeichert werden. Der Zugriff auf beliebige Stellen der Platte ist recht schnell.

Diskette Bei Kleinrechnern handelt es sich häufig um flexible Platten, sogenannte Disketten.

Maschinensprache Die CPU verkehrt über Datenleitungen mit den Speichern und externen Geräten nur mit binär verschlüsselten Daten. Die Elektronik kennt nämlich nur die Zustände An und Aus, d. h. nur zwei Ziffern. Daher also die Arbeit im Binärsystem (Zweiersystem). Das Programmieren im Binärsystem (d. h. auf Maschinenebene) ist recht aufwendig und unkomfortabel. Eine simple Multiplikation macht schon Schwierigkeiten.

Höhere Programmiersprachen So hat man schon sehr früh die sogenannten höheren Programmiersprachen entwickelt. Hier wird die Multiplikation z. B. durch die Anweisung 6*7 ausgeführt. Eine der höheren und komfortableren Programmiersprachen ist Pascal. Nachdem wir diese Sprache gelernt haben, können wir dem Computer mit ziemlich einfachen und verständlichen Anweisungen klarmachen, was wir von ihm wollen.

Compiler Ein sogenannter Compiler (Übersetzungsprogramm) sorgt dann dafür, daß unsere Anweisungen in einen für die Maschine verständlichen Code übersetzt werden.

Algorithmus

1.3 Problemanalyse – Planung von Programmen

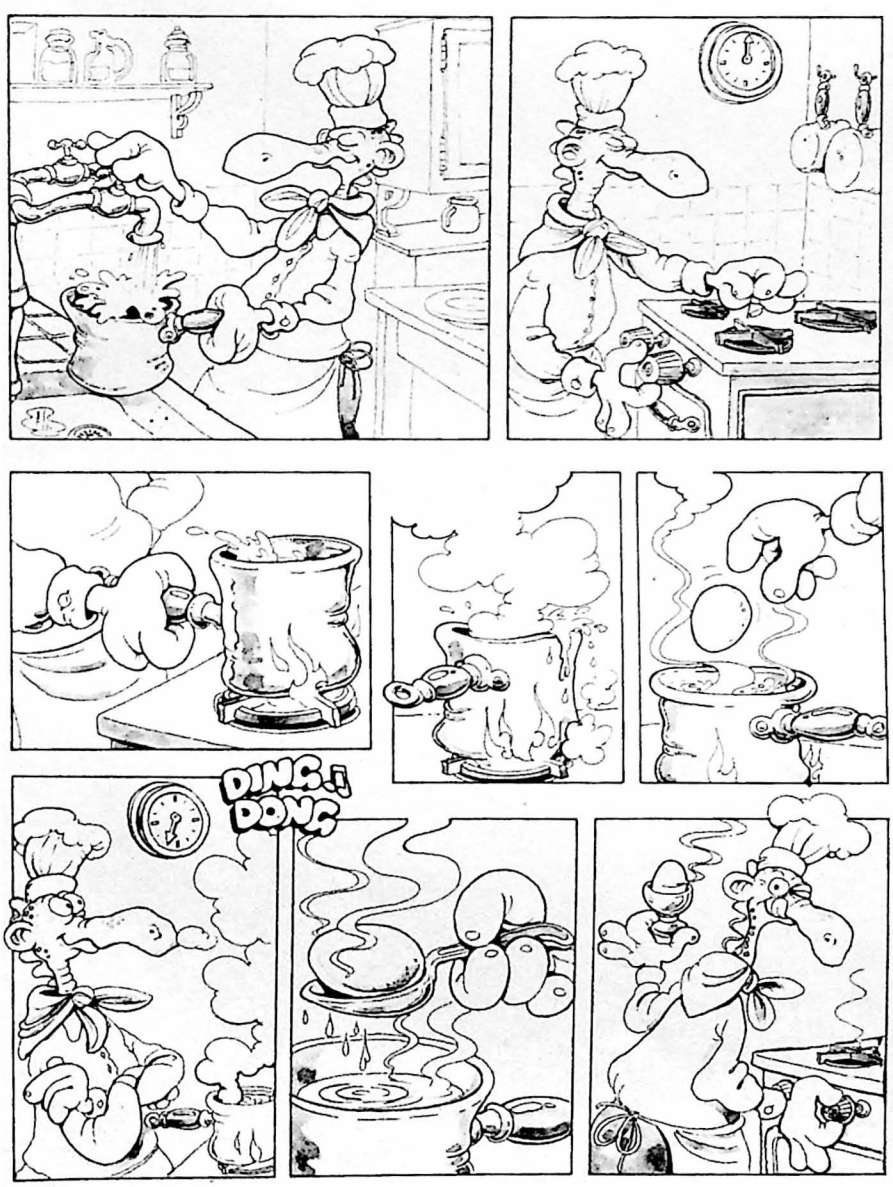
Programm/ Anweisungen

Der Computer wird verwandt, um Probleme zu lösen. Da der Computer keine eigene Intelligenz besitzt, sondern nur Befehle (Anweisungen) ausführen kann, ist es an uns, eine Reihe von Anweisungen (ein Programm) aufzustellen, die zur Problemlösung angemessen sind.

Algorithmus

Bevor wir überhaupt ein Programm planen, sollten wir wissen, was wir überhaupt wollen. Dazu ist es nötig, einen möglichst exakten Anforderungskatalog für die jeweilige Problemstellung zu erstellen. Danach wird ein geeigneter Algorithmus gesucht. Ein Algorithmus ist eine endliche, exakte und korrekte Vorschrift, die bei geeigneten Eingabedaten zwangsläufig zu korrekten Ausgabedaten führt.

Ein Beispiel für einen Alltagsalgorithmus:



Flußdiagramm

Variable Eine Variable ist eine veränderliche Größe, die eine Platzhalter-Funktion hat und der ein oder mehrere Werte zugewiesen werden können.

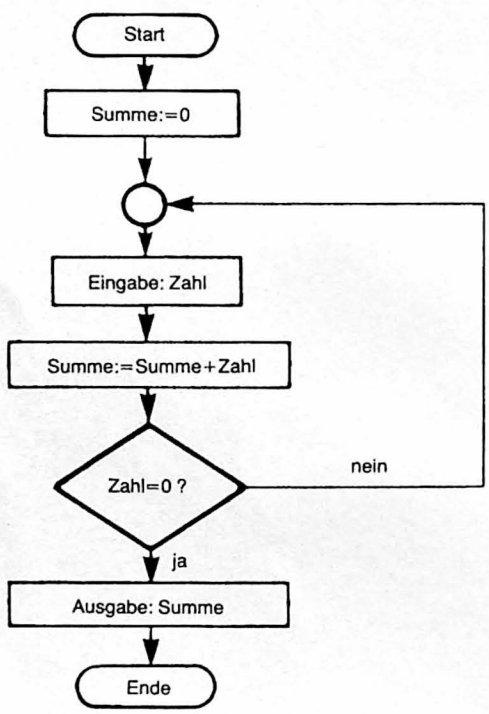
Beispiel Aufgabe des Programms: Zahlen addieren und die Summe ausgeben.

In Einzelschritten heißt das:

Anforderungen
Setze die Variable Summe auf Null.
Wiederhole: Einlesen einer Zahl;
addiere die Zahl zu Summe;
bis die eingegebene Zahl gleich Null ist.
Gebe die Summe der Zahlen aus.

Flußdiagramm Die Planung des Programms erfolgt erst einmal auf dem Papier. Dazu bedient man sich häufig sogenannter Flußdiagramme oder Blockdiagramme.

Flußdiagramm zu unserem Programm:



Diese Art der schematischen Planung eignet sich für Pascal-Programme nicht so gut, da die Pfeile in Flußdiagrammen für Sprünge an andere Stellen im Programm stehen.

Blockdiagramm Pascal jedoch ist eine blockorientierte Sprache, in der man ganz ohne Sprünge auskommen kann (in diesem Buch wird auf Sprünge ganz verzichtet). Wir benutzen daher besser ein Blockdiagramm:

Blockdiagramm

Programm Test
Variablen Zahl, Summe : Dezimalzahl
Anfang
Summe := 0
Wiederhole
Eingabe: Zahl
Summe := Summe + Zahl
bis Zahl=0
Ausgabe: Summe
Ende.

In vielen Programmplanungen in diesem Buch werden wir uns mit dieser Struktur anfreunden.

Die zu verwendenden Symbole für Block- und Flußdiagramme sind im Anhang aufgeführt. ***Symbole***

Wenn die Planung abgeschlossen ist, übersetzen wir die Anweisungen unseres Blockdiagramms in Pascal:

Beispiel-Programm

```
PROGRAM Test;

VAR Zahl, Summe : REAL;

BEGIN
  Summe := 0;
  REPEAT
    READLN (Zahl);
    Summe := Summe + Zahl
  UNTIL Zahl=0;
  WRITELN (Summe:7:2)
END.
```

Der Vorteil eines Blockdiagramms liegt auch darin, daß man stufenweise die Planung verfeinern kann (eine Technik, auf die die Programmiersprache Pascal sehr gut zugeschnitten ist). ***Verfeinerung***

So ließe sich ein Adreßverwaltungsprogramm etwa folgendermaßen grob planen: ***Beispiel***

Programm Adressen				
Variablen ch : Zeichen <andere fehlen noch>				
Anfang				
Menue				
Wiederhole				
Eingabe: ch				
falls ch ist:				
'E'	'A'	'D'	'S'	'L'
Eingabe	Alpha	Drucke	Speichere	Lies
bis ch= 'Z'				
Ende.				

Die Planung enthält einige noch in gleicher Weise zu planende Blöcke (Unterprogramme):

Menue

Menue: Zeigt dem Benutzer die verschiedenen Möglichkeiten:
E)ingabe
A)lphabetisch Sortieren
D)rucken
S)peichern auf Diskette
L)esen von Diskette
Z)um Schluß

Unterprogramme

Eingabe: Eingeben und Verändern von Adressen
Alpha: Sortieren der Adressen
Drucke: Drucken der Adressen
Speichere: Speichern der Daten auf Diskette
Lies: Lesen der Daten von Diskette

Natürlich handelt es sich hier um eine recht grobe Planung, was allein schon an den fehlenden Variablen zu sehen ist. Außerdem muß hier noch überlegt werden, wie die Variablen von einem Unterprogramm zu einem anderen übergeben werden.

Fehler

Häufig schleichen sich bei Planung und Eingabe Fehler ein. Diese zu finden und sinnvoll zu berichtigen, braucht oft viel Zeit und einige Erfahrung bei großen Programmen.

→ Daher: Viel am Rechner programmieren!

Dokumentation

Schon beim Erstellen des Programms, spätestens jedoch danach wird das Programm dokumentiert. Dies geschieht durch Kommentare im Programmtext sowie durch Begleittexte über Planung, verwendete Variablen und Funktion des Programms. Die Dokumentation ist sehr wichtig, um das Programm noch nach einiger Zeit zu verstehen und eventuell zu verändern und zu berichtigen.

Keine Sofortprogramme

Auf keinen Fall sollten Sofortprogramme erstellt werden, es sei denn, das Problem ist sehr klein und übersichtlich. Programme ohne Planung sind fast immer umständlich und kaum lesbar.

Übungen:

1.3.1 Kann man einen Algorithmus „Auto fahren" aufstellen?

1.3.2 Beschreiben Sie einen Algorithmus für das Problem: „Schreiben und Absenden eines Briefs".

1.3.3 Beschreiben Sie einen Algorithmus für das Problem: „Telefonieren an einem Münzfernsprecher".

Erste Schritte mit dem UCSD-Pascal-Betriebssystem

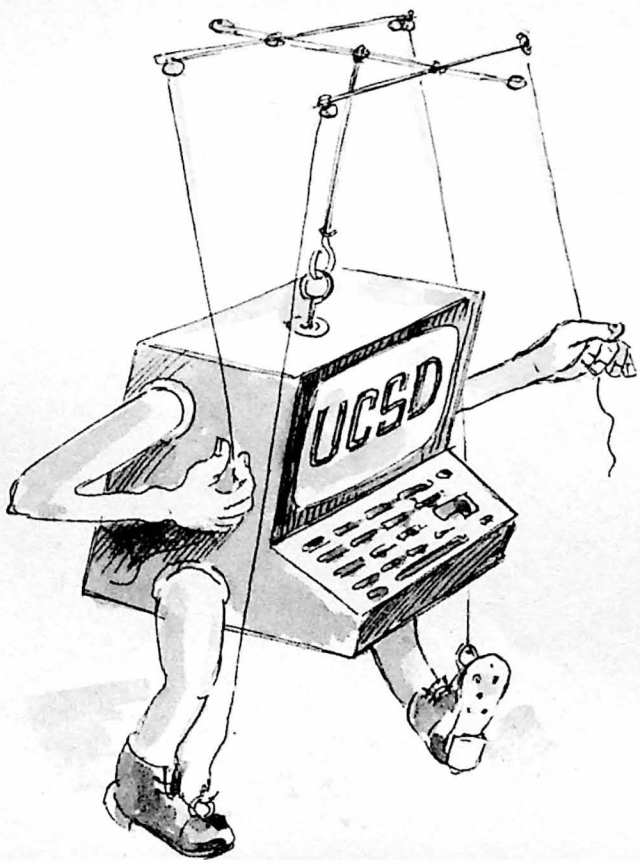
Kapitel 2

In diesem Kapitel beschäftigen wir uns mit dem UCSD-Betriebssystem, einem der Betriebssysteme, auf denen die Programmiersprache Pascal läuft. Das Betriebssystem ermöglicht uns, Texte (z. B. Programmtexte) zu schreiben, diese Texte in lauffähige Programme zu übersetzen und mit Disketten zu hantieren. Es werden nur die wichtigsten Befehle des Betriebssystems behandelt und an Beispielen erklärt, da der Anfänger erst allmählich einen Bedarf für alle (oft recht komfortablen) Möglichkeiten entwickelt.

Eine Zusammenfassung aller Befehle des Betriebssystems findet sich im Anhang.

Nach dem Lesen dieses Kapitels

- wissen wir, wie das System gestartet wird;
- können wir Disketten formatieren und kopieren;
- wissen wir, was Volumes und Files sind;
- können wir mit dem Editor Texte (auch Programmtexte) schreiben und verändern;
- haben wir ein erstes Programm getestet.



2.1 Wir fangen an

Start:

„Booten" Der Start des Pascal-Systems hängt immer ein wenig vom benutzten Rechner ab. In Zweifelsfällen sollte in der Bedienungsanleitung nachgesehen werden, wie das System „gebootet" (gestartet) wird.

2-Laufwerke-Start Beim APPLE II mit zwei Diskettenlaufwerken haben wir folgendes zu tun: In Laufwerk #4: (Drive 1) wird die Diskette namens APPLE1: und in Laufwerk #5: (Drive 2) die Diskette namens APPLE2: eingelegt, und der Rechner wird eingeschaltet.

Haupt-Kommandozeile Nach dem Einschalten meldet sich der Rechner mit einem Begrüßungstext, der Informationen über die verwendete Pascal-Version enthält. Außerdem finden wir am oberen Bildschirmrand die sogenannte COMMAND-Zeile (Hauptkommandozeile):

```
Command: E(dit,R(un,F(ile,C(omp,L(ink,X(ecute,A(ssem,D(ebug, ?
(1.1)
```

Dies sind die Abkürzungen für die Systembefehle. Drückt man einen der ersten Buchstaben, so wird der entsprechende Befehl ausgeführt. Wird auf die Fragezeichen-Taste gedrückt, so erscheint eine weitere Kommandozeile, der 2. Teil.

Bedeutung der Befehle Die Bedeutung aller Befehle ist im Anhang zu finden. Hier nur die wichtigsten:

F(ile: Der FILER wird aufgerufen, das System, mit dem die Disketteninhalte angesehen, Programme, Daten oder ganze Disketten kopiert werden können.

E(dit: Der EDITOR wird aufgerufen. Mit dem EDITOR lassen sich (Programm-)Texte erstellen, verändern und ansehen.

R(un: Mit diesem Befehl wird das Programm gestartet, das unter dem Namen SYSTEM.WRK.TEXT gespeichert ist, das sogenannte Workfile (Arbeitsprogramm).

X(ecute: Diesen Befehl benutzen wir, um ein beliebiges Programm (das natürlich als Codefile, d. h. übersetzt vorliegen muß) ausführen.
Der Rechner fragt nach dem Namen des Programms.

Disketten-Inhalt *Was ist auf der Diskette?*

Wir sehen nach, was auf unseren Disketten zu finden ist.

Filer Dazu rufen wir aus der Hauptkommandozeile den Filer auf, indem wir F drücken. Eine neue Kommandozeile erscheint:

```
Filer: G(et,S(ave,N(ew,L(dir,R(em,C(hng,T(rans,D(ate,Q(uit
```

Files/Volumes

Wir drücken L für List Directory und werden gefragt:

Directory/ Inhaltsverzeichnis

Dir listing of what vol?:
(Inhaltsverzeichnis welcher Diskette?)

Wir antworten mit der Nummer des Laufwerks, in der die zu untersuchende Diskette liegt (z. B. #4: für Laufwerk 1), und drücken <Return>.

```
APPLE1:
SYSTEM.APPLE          32     26-JUL -79
SYSTEM.PASCAL         36      4-MAY-79
SYSTEM.MISCINFO        1      4-MAY-79
SYSTEM.EDITOR         45     29-JAN -79
SYSTEM.FILER          28     24-MAY-79
SYSTEM.LIBRARY        36     22-JUN -79
SYSTEM.CHARSET         2     14-JUN -79
SYSTEM.SYNTAX         14     18-APR-79
8/8 FILES, 80 UNUSED, 80 IN LARGEST
```

Dies könnte eine Ausgabe des Inhaltsverzeichnisses (Directory) der Diskette sein.

Durch Drücken von Q für Q(uit (Verlassen) kommen wir wieder aus dem Filer heraus zur Hauptkommandozeile.

Quit

Konzept (Files/Volumes):

Das UCSD-Pascal-System unterscheidet Files und Volumes.

Volumes: Volumes sind an den Rechner angeschlossene „Geräte" (im weitesten Sinne). Dies sind die Disketten, der Drucker, der Bildschirm, die Tastatur und andere mögliche Geräte.

Volumes („Geräte")

Alle Volumes haben Namen und Nummern:

Namen/ Nummern

Nummer:	Name:	Bedeutung:
#1:	CONSOLE:	Bildschirm und Tastatur
#2:	SYSTERM:	dito ohne Ausgabe auf Bildschirm
#4:	<diskettenname>:	Diskette in Laufwerk 1
#5:	<diskettenname>:	Diskette in Laufwerk 2
#6:	PRINTER:	Drucker

Fehlende Nummern sind entweder noch nicht belegt oder Erweiterungen, die an dieser Stelle belanglos sind.

→ *Wichtig:* Jedes Volume kann durch seine Nummer oder seinen Namen angesprochen werden. Beides muß unbedingt durch einen Doppelpunkt abgeschlossen werden, um es als Volume kenntlich zu machen.

Files: Einige Volumes, nämlich Disketten, werden weiter unterteilt in sogenannte Files (Dateien).

Files (Dateien)

Namen/ Typen

Folgende Arten von Files finden wir auf Disketten:

SYSTEM.name	– Systemprogramm (EDITOR,COMPILER usw.)
name.TEXT	– Textfile, d. h. ein lesbarer Programmtext
name.CODE	– Codefile, d. h. übersetztes Programm
name.DATA	– Daten (z. B. Zahlen, Texte usw.)
name.FOTO	– ein Grafik-Bild
SYSTEM.WRK.TEXT	– aktuell bearbeiteter Programmtext
SYSTEM.WRK.CODE	– aktuell übersetztes Programm

Workfile

Etwas Besonderes hat es mit den sogenannten Workfiles auf sich: Wenn wir ein neues Programm schreiben, erstellen wir zuerst ein sogenanntes Workfile (ein File ist ein Datensatz, d. h. Programmtext, übersetztes Programm, Daten oder Systemprogramm). Dieses Workfile ist also das vom Rechner aktuell bearbeitete Programm. Das Workfile muß für den Rechner erreichbar sein. Es wird auf der Diskette mit dem Namen SYSTEM.WRK.TEXT abgespeichert. Der Anhang .TEXT bedeutet, daß es sich um einen lesbaren Programmtext handelt.

Wenn das Programm übersetzt ist, wird es als SYSTEM.WRK.CODE auf der Diskette gespeichert. Der Anhang .CODE bedeutet, daß es sich um ein übersetztes, d. h. ausführbares Programm handelt.

Back-up

Kopieren von Disketten:

Von wertvollen Disketten (z. B. den Systemdisketten) erstellen wir Kopien (Back-ups).

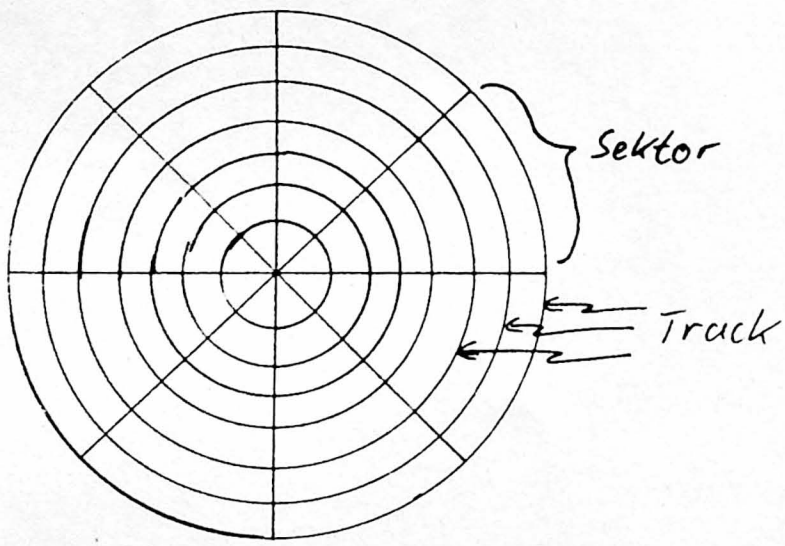
Formatieren

Formatieren:

Zuerst müssen wir neue Disketten formatieren. Warum?

Track/Sektor

Eine gekaufte Diskette enthält vorerst nichts. Sie ist für den Rechner nicht lesbar, da er keine Anhaltspunkte hat, wo auf der Diskette etwas zu finden ist. Beim Formatieren wird nun die Diskette in sogenannte Tracks (Spuren) und Sektoren eingeteilt:



Disketten kopieren

Wir legen die Diskette mit dem Formatierungsprogramm (bei APPLE: APPLE3:FORMATTER.CODE) in Laufwerk #5: und wählen aus der Hauptkommandozeile X für execute. Auf die Frage nach dem auszuführenden Programm antworten wir mit: #5:<programmname> (z. B.: #5:FORMATTER). Nach dem Herausnehmen der Diskette aus Laufwerk #5: werden nun in diesem Laufwerk alle Disketten formatiert. Zum Abschluß wird auf die Frage nach dem Laufwerk nur <Return> gedrückt.

FORMATTER

→ Frisch formatierte Disketten erhalten den Namen „BLANK:".

Kopieren:

Kopieren

Nun drücken wir wieder F für Filer. Im Filer wählen wir T für T(ransfer (übertragen). Hier fragt der Rechner, welches File (Datensatz) übertragen werden soll. Um eine ganze Diskette zu übertragen, nennen wir den Namen der Diskette (WICHTIG: mit Doppelpunkt). Danach fragt der Rechner, wohin übertragen werden soll, und wir antworten mit dem Namen der Leerdiskette.

Kurzform

Dieser Dialog geht noch einfacher, wenn wir gleich Original- und Zieldiskette durch Komma getrennt nennen. Bei der Frage nach den Files geben wir z. B. ein:

APPLE1:,BLANK: , wenn wir APPLE1 auf BLANK kopieren wollen.

Wenn zwei Laufwerke zur Verfügung stehen, können auch die Nummern der Laufwerke genannt werden, z. B.:

#4:,#5: , wenn die Diskette in Nummer 4 auf die Diskette in Nummer 5 kopiert werden soll.

Nun fragt der Rechner, ob 280 Blöcke (d. h. alle Blöcke) kopiert werden sollen, und wir antworten mit Y für Yes. Als nächstes will der Rechner noch wissen, ob wirklich der Inhalt der Diskette BLANK: zerstört, d. h. überschrieben werden soll – eine sehr nützliche Vorsichtsmaßnahme. Wir antworten natürlich mit Y.

Übungen:

2.1.1 Kopieren Sie Ihre Systemdisketten.

2.1.2 Sehen Sie sich die Inhaltsverzeichnisse (Directories) Ihrer Disketten an.

2.1.3 Kopieren Sie einzelne Files.

2.1.4 Schreiben Sie den Vorgang des Kopierens als Blockdiagramm.

2.1.5 Wie unterscheiden sich Magnetband und Diskette?

2.1.6 Was heißt Formatieren? – Schreiben Sie den Vorgang des Formatierens als Blockdiagramm.

Mit Hilfe des Anhangs:

2.1.7 Bringen Sie die Inhaltsverzeichnisse Ihrer Disketten auf den Drucker.

2.1.8 Ändern Sie das Datum.

2.1.9 Benennen Sie Files (Disketten) um.

2.1.10 Löschen Sie Files.

2.2 Unser erstes Programm

Texteingabe im Editor

Programmtexte (wie auch andere Texte) werden stets mit Hilfe des Editors geschrieben. Sie werden als Textfiles abgespeichert. Anschließend werden Programmtexte mit dem Compiler übersetzt und sind als Codefiles lauffähige Programme.

Wir schreiben nun ein kleines Programm, das nach dem Namen des Benutzers fragt und ihn anschließend mit seinem Namen begrüßt.

Wir drücken E in der Hauptkommandozeile für den Editor. Es erscheint:

```
>Edit:
No workfile is present. File? (<ret> for no file <esc-ret> to exit)
:■
```

Neues Workfile

Es gibt noch kein Workfile (d. h. keine Arbeitsdatei), daher drücken wir <Return>. Nun haben wir die Kommandozeile des Editors vor uns:

```
>Edit: A(djust C(opy D(elete F(ind I(nsert J(ump R(eplace
Q(uit X(chng Z(ap
```

Insert

Mit I gelangen wir in den Insert-Modus. Wir schreiben nun die folgenden Zeilen (<Return> : Drücken der Return-Taste ; <L> : Leertaste ; <B> : Backspace-Taste (Pfeil links)):

Eingabe

```
PROGRAM ERSTES; <Return>
<Return>
VAR NAME : STRING; <Return>
BEGIN <Return>
<L> <L> WRITE ('NAME'); <Return>
READLN (NAME); <Return>
WRITELN <Return>
WRITELN ('GUTEN TAG ',NAME) <Return>
<B> <B> END.
```

Fehlerberichtigung

Auf dem Bildschirm muß nun zu sehen sein:

Programmtext

```
PROGRAM ERSTES;

VAR NAME : STRING;
BEGIN
   WRITE ('NAME');
   READLN (NAME);
   WRITELN
   WRITELN ('GUTEN TAG ',NAME)
END.
```

Nun drücken wir die <etx>-Taste (beim APPLE ist das die Taste C mit gleichzeitig gedrückter CTRL-Taste).

Damit wird der Text angenommen.

Mit Q verlassen wir den Editor. Es erscheint: ***Quit***

```
>Quit:
      U(pdate the workfile and leave
      E(xit without updating
      R(eturn to the editor without updating
      W(rite to a filename and return
      S(ave with same name and return
```

Wir drücken U zum Erstellen eines neuen Workfiles. Wieder erscheint die Hauptkommandozeile. Hier wird nun R für Run gedrückt. Das System ruft den Compiler auf und beginnt mit der Übersetzung des Programms. ***Update***

Leider erscheint eine Fehlermeldung! ***Fehler***

Wir geben E für Rückkehr in den Editor ein. Hier meldet der Rechner, daß wahrscheinlich in der vorhergehenden Zeile ein Semikolon fehlt. Wie recht er hat! Denn jede Anweisung (Programmzeile) in Pascal wird mit einem Semikolon abgeschlossen. Vor END braucht kein Semikolon zu stehen.

Nach dem Drücken der Leertaste gehen wir mit den Tasten Pfeil-rechts und Pfeil-links an die entsprechende Stelle ***Fehlerberichtigung***

```
READLN (NAME);
WRITELN■
```

und fügen durch die Tastenfolge: I für Insert, ; , <etx> (CTRL-C für Annehmen) das Semikolon ein.

Editor-Befehle

Der neue Programmtext:

Fertiges Programm

```
PROGRAM ERSTES;

VAR NAME : STRING;
BEGIN
   WRITE ('NAME');
   READLN (NAME);
   WRITELN;
   WRITELN ('GUTEN TAG ',NAME)
END.
```

Run Wieder verlassen wir wie oben den Editor (mit neuem UPDATE) und lassen das Programm mit R laufen.

Nun funktioniert es!

Abspeichern des Programms Wir speichern das Programm auf einer Diskette (z. B. mit Namen BLANK:) unter dem Namen „Prog1" ab.

Dazu rufen wir mit F den Filer auf und wählen S für Save (Speichern). Der Rechner fragt: „Save as what file?", und wir antworten mit BLANK:PROG1 <Return>. Sofern die Diskette BLANK: in einem der Laufwerke liegt, wird das Programm (Text- und Codefile) gespeichert. Nun löschen wir das Workfile mit N für New und verlassen mit Q für Quit den Filer.

2.3 Wir schreiben einen Brief mit dem Editor

In diesem Paragraphen schreiben wir einen kleinen Bericht für die Schülerzeitung über unsere ersten Eindrücke vom Informatikunterricht.

Datum setzen Dazu starten wir das System (wie in Kap. 2.1 beschrieben), wählen F für Filer aus der COMMAND-Zeile und setzen mit D für Date das aktuelle Datum (z. B. 10-OCT-83 <Return>).

Workfile löschen Damit wir sicher mit einem neuen Text anfangen können, löschen wir mit N für New das Workfile und gehen mit Q für Quit in die COMMAND-Ebene zurück.

Mit E wählen wir den Editor aus. Es erscheint:

```
>Edit: A(djust C(opy D(elete F(ind I(nsert J(ump R(eplace Q(uit
X(chng Z(ap
```

Anhang Im Anhang findet sich eine Übersicht über die Editor-Anweisungen.

Insert Wir wählen I für Insert (Einfügen) und fangen mit dem Schreiben unseres Textes an. Am Ende einer jeden Zeile drücken wir <Return>, um in die nächste Zeile zu gelangen:

Editor-Befehle

Text

Meine ersten Eindruecke vom Fach Informatik:

Zuerst haben wir den Rechner kennengelernt. Dieser besteht aus einem Gehaeuse mit einer Schreibmaschinentastatur, einem Bildschirm und zwei Diskettenlaufwerken. Die Disketten werden zum Abspeichern von Programmen und Daten verwendet. Es handelt sich dabei um runde, magnetisch beschichtete Scheiben, auf denen wie auf einem Tonband magnetische Spuren geschrieben werden. Der Bildschirm wird fuer die Ausgaben, die Tastatur fuer die Eingaben verwendet.

Als der Rechner eingeschaltet wurde, hat er uns erst einmal freundlich begruesst. Doch dann wurde es etwas komplizierter. Wir mussten uns einige Befehle merken, um mit dem Rechner zu kommunizieren. Ich nehme aber an, daß man sich im Laufe der Zeit an diese Befehle gewoehnen wird.

Der Rechner scheint sehr penibel zu seim. Er reagiert sofort auf jeden Fehler und laesst keine Ungenauigkeiten bei Befehlen zu.

CTRL-C

Nachdem dieser Text soweit getippt ist, drücken wir Ctrl-C, und der Text ist vom Rechner angenommen. Die Edit-Zeile erscheint wieder.

Jump

Mit J für Jump (Sprung), gefolgt von B und E, springen wir an den Anfang (B) und wieder an das Ende (E) des Textes.

eXchange

Da, ein Tippfehler! In der vorletzten Zeile muß es „sein" statt „seim" heißen.

Cursorbewegung

Mit der <Pfeil-links>-Taste bewegen wir den Cursor auf das zu verbessernde m, tippen X für eXchange (Austausch), berichtigen in n und drücken Ctrl-C.

zeilenweise vor und zurück

Nun gehen wir zeilenweise weiter, indem wir die <Return>-Taste drücken. Am Ende angekommen drücken wir < (damit ändern wir die Bewegungsrichtung) und gehen zeilenweise mit der <Return>-Taste an den Anfang. Hier drehen wir die Bewegungsrichtung mit > wieder um.

Replace

Statt „Rechner" soll es nun im ganzen Text „Computer" heißen. Dazu drücken wir / (für „alle") und R für Replace (Ersetze). Nun wird eingegeben /Rechner//Computer/ <Return>, und das Wort „Rechner" wird im ganzen Text in „Computer" umgewandelt.

Delete

Wir wollen per Hand das Wort „Ungenauigkeiten" durch „Unregelmaessigkeiten" ersetzen. Dazu bewegen wir den Cursor an den Anfang des Wortes und löschen es mit D für Delete und Bewegung des Cursors über das Wort mit der <Pfeil-rechts>-Taste. Das Löschen wird mit Ctrl-C abgeschlossen. Nun wird mit I für Insert das neue Wort eingegeben und die Eingabe mit Ctrl-C abgeschlossen.

Editor-Befehle

Adjust

Aufgabe: Experimentieren Sie mit dem A(djust-Befehl (siehe Anhang).

Quit

Wir verlassen den Editor nun mit Q für Quit und speichern den Text als neues Workfile mit U für UPDATE.

Text → Drucker

Nun bringen wir den Text auf den Drucker.

Dazu wählen wir mit F den Filer und hier mit T den Transfer-Befehl. Auf die Frage „Transfer what file?" antworten wir mit:

#4:SYSTEM.WRK.TEXT,#6: <Return>

. . . und der Text wird ausgedruckt.

Sollte kein Drucker (#6:) vorhanden sein, so simulieren wir mit dem Bildschirm (#1:) den Drucker.

Übungen:

2.3.1 Speichern Sie den Text auf einer anderen Diskette ab.

2.3.2 Wie kommt man auf möglichst einfache Weise an Anfang und Ende eines Textes?

2.3.3 Schreiben Sie einen Text und kopieren Sie ihn auf möglichst einfache Weise 10mal. (Benutzen Sie dazu den Copy-Befehl (siehe Anhang).

2.3.4 Nehmen Sie Stellung zu der Aussage: Beim R(eplace-Befehl ist auf die Länge des neuen Wortes zu achten.

2.3.5 Schreiben Sie sich „Entschuldigungsformulare" für verschiedene Anlässe (Unwohlsein, Mangel an Verständnis, Goldhamster gestorben usw.), und drucken Sie diese.

2.4 Zusammenfassung

Was haben wir erreicht?

Wir kennen nun die Bedeutung einiger Anweisungen der Hauptkommandozeile sowie der Kommandozeilen des Editors und Filers. Außerdem sind wir in der Lage, die wichtigsten Befehle im Editor und im Filer korrekt zu gebrauchen. Sicher werden wir noch das ein oder andere Mal in diesem Kapitel und im Anhang nachsehen, wenn wir eine Anweisung länger nicht benutzt haben. Versuchen Sie aber, möglichst viele der bekannten Anweisungen noch oft zu gebrauchen und sie einzuüben.

Weiterhin haben wir gelernt, was es mit Files und Volumes im UCSD-Pascal auf sich hat, und wir kennen die spezielle Bedeutung des Workfiles. Wir können nun Disketten formatieren, duplizieren und, wenn wir genau aufgepaßt haben, auch Files von den Disketten kopieren.

Selbst ein kleines Programm haben wir erfolgreich erstellt und „gefahren".

Konzept und Aufbau eines Pascal-Programms

Kapitel 3

In diesem Kapitel wird dargestellt, nach welchen Grundprinzipien ein jedes Pascal-Programm und seine Teile aufgebaut sind. Außerdem erfährt der Leser hier einfache Ein- und Ausgabebefehle, damit es möglich wird, kleine Programme zu schreiben.

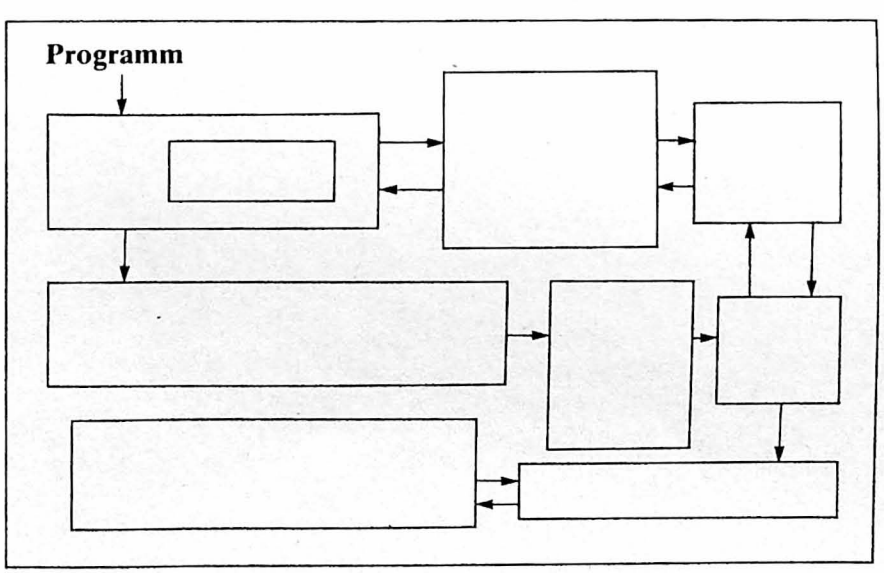
Der Deklarationsteil eines Programms wird vorgestellt als der Teil, in dem für das Programm und seine Teile geklärt wird, welche Variablen und Konstanten benutzt werden. Dazu werden einige einfache Variablentypen dargestellt und an Beispielen verdeutlicht.

Zur Anwendung des Gelernten werden einige Beispielprogramme erstellt, an denen auch die im Pascal vorhandenen arithmetischen Operationen vorgestellt werden.

Mit der Darstellung einiger Grafikbefehle wird die Voraussetzung für Grafikanwendungen in Übungen geschaffen.

Nach dem Lesen dieses Kapitels . . .

- kennen wir die Blockstruktur von Pascal-Programmen (BEGIN...END) und können sie anwenden;
- kennen wir die einfachen Ein- und Ausgabebefehle READ, READLN, WRITE, WRITELN und können sie anwenden;
- können wir zwischen Konstanten und Variablen unterscheiden;



- kennen wir den Aufbau eines Pascal-Programms mit folgenden Teilen: Titel, Deklarationsteil, Programmrumpf;
- wissen wir, wie man durch den Variablentyp Interactive den Drucker innerhalb des Programms anwenden kann;
- kennen wir die in Pascal möglichen arithmetischen Operationen;
- können wir kleine Rechen- und Grafikprogramme erstellen.

3.1 Das Grundprinzip eines Pascal-Programms und seiner Teile

Struktur

Man hört häufig, daß Pascal eine strukturierte Programmiersprache sei. Was ist damit gemeint?

Das ganze Pascal-Programm hat eine bestimmte Struktur, auf die wir gleich zu sprechen kommen. Außerdem hat jeder Teil des Programms eine sehr ähnliche Struktur. Die Strukturen, um die es sich hier handelt, werden oft Blöcke genannt (siehe Blockdiagramme) und sind so geartet, daß sie eine recht einfache Lösung eines Programmierproblems zulassen.

Hier ein Beispiel eines Pascal-Programms:

(Das Programm ist recht umfangreich, damit möglichst einige verschiedene Strukturen sichtbar werden – es ist sicher an dieser Stelle noch nicht zu verstehen und kann eventuell als Übung für den Umgang mit dem Editor eingetippt und gestartet werden.)

Programmkopf

Deklarationsteil

```
PROGRAM Vielecke;
  (* Beispielprogramm mit Prozeduren - fuer grafikfaehige Rechner *)

  USES TURTLEGRAPHICS;    (* ruft die Grafikbibliothek auf *)

  VAR Frage:CHAR;         (* fuer die Frage nach weiterem Vieleck *)

  PROCEDURE Rahmen;       (* zeichnet Rahmen auf Bildschirm *)
    BEGIN
      PENCOLOR(NONE);     (* ohne Farbe zum Ursprung *)
      MOVETO(0,0);
      PENCOLOR(WHITE);    (* mit Farbe zu allen Ecken *)
      MOVETO(279,0);
      MOVETO(279,191);
      MOVETO(0,191);
      MOVETO(0,0)
    END; (* von Rahmen *)


  PROCEDURE Vieleck;      (* zeichnet Vieleck *)
    CONST MaxN = 25;      (* maximale Eckenzahl *)
    VAR X, Y, Laenge, Winkel, I, N : INTEGER;

    BEGIN
      TEXTMODE;           (* auf Textbildschirm schalten *)
      WRITE ('Eingabe der Eckenzahl: ');
      READLN (N);
      IF N>MaxN THEN BEGIN
          WRITELN ('zu viele...');
          WRITELN ( MaxN , ' Ecken !');
          N := MaxN
        END; (* von if *)
      WRITE ('Eingabe Seitenlaenge: ');
      READLN (Laenge);
      WRITE ('Anfang bei X= ');
      READLN (X);
      WRITE ('   und bei Y= ');
      READLN (Y);
      Winkel := 360 DIV N;
```

(Fortsetzung des Programms)

```
      GRAFMODE;              (* auf Grafikbildschirm schalten *)
      PENCOLOR(NONE);        (* ohne Farbe zum Anfangspunkt *)
      MOVETO(X,Y);
      PENCOLOR(WHITE);       (* mit Farbe Vieleck zeichnen *)
      FOR I:=1 TO N DO BEGIN
         MOVE(Laenge);
         TURN(Winkel)
       END;(* von for *)
      READLN
    END; (* von Vieleck *)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    BEGIN                    (*HAUPTPROGRAMM*)
      WRITELN ('Dieses Programm zeichnet Vielecke');
      WRITELN ('weiter mit RETURN');
      INITTURTLE;            (* Grafik initialisieren *)
      VIEWPORT(0,279,0,191);(* Bildschirmfenster setzen *)
      Rahmen;                (* Prozedur Rahmen aufrufen *)
      REPEAT                 (* Wiederhole...bis Frage='N' *)
        Vieleck;             (* Prozedur Vieleck aufrufen *)
        TEXTMODE;
        WRITE ('Wollen Sie noch ein N-Eck (J/N)?');
        READ (Frage);
        WRITELN
      UNTIL Frage='N'
    END.  (* Programmende *)
```

Hauptprogramm

Das Programm Vielecke bewirkt bei grafikfähigen Rechnern, daß ein regelmäßiges N-Eck mit vorgegebener Eckenzahl und Seitenlänge an die Stelle (X,Y) gezeichnet wird. Die Funktion des Programms ist größtenteils im Programmtext selber durch eingefügte Kommentare erklärt.

Kommentar

→ Kommentar wird in geschweiften Klammern geschrieben. Da viele Rechner diese Zeichen nicht darstellen können, wird oftmals als Ersatzzeichen (* Kommentar *) verwendet.

Hier sollen die Befehlsfolgen des Programms nicht im einzelnen erklärt werden. Dies wird in den folgenden Kapiteln geschehen. Vielmehr wollen wir uns seine Struktur ansehen.

Wir teilen das Programm in drei große Abschnitte ein:

Teile des Programms

1. Programmkopf
2. Deklarationsteil (mit Prozeduren/Funktionen)
3. Hauptprogramm

Zu 1.: Der Programmkopf besteht aus dem reservierten Wort PROGRAM und dem Programmnamen.

1. Programmkopf

Zu 2.: Als erstes enthält der Deklarationsteil (wenn nötig) eine USES-Deklaration, die angibt, welche Bibliotheksteile (UNITs) verwendet werden (eine Bibliothek ist eine Sammlung schon fertiger Programme). Dann werden Konstanten, Datentypen und Variablen vereinbart, die im Programm vorkommen und gebraucht werden.

2. Deklarationsteil

Weiterhin folgen alle Prozeduren und Funktionen des Programms. Prozeduren sind Unterprogramme, die vom Hauptprogramm aufgerufen und dann abgearbeitet werden. Die Prozeduren haben die gleiche Form wie ein Hauptprogramm selbst, d. h. sie bestehen aus dem Prozedurkopf mit Deklarationsteil, evtl. aus weiteren Prozeduren und dem Prozedurrumpf mit den Befehlen zwischen BEGIN und END.

Prozeduren/ Funktionen

Außer den Prozeduren können noch Funktionen im Programm enthalten sein, die sich etwas in Form und Gebrauch von Prozeduren unterscheiden (siehe Kap. 9 und Kap. 10).

3. Hauptprogramm

Zu 3.: Das Hauptprogramm fängt mit BEGIN an und hört mit END auf. Es besteht aus einzelnen Anweisungen und Blöcken von Anweisungen, die jeweils durch BEGIN und END zusammengehalten werden.

GOTO

In manchen Programmiersprachen werden die Befehle in der Reihenfolge der Zeilennummern (z. B. in BASIC) oder durch Sprünge zu bestimmten Zeilennummern abgearbeitet.

Anders in Pascal. Hier werden Befehle immer in der beschriebenen Reihenfolge abgearbeitet – für BASIC-Anwender erst ein wenig unverständlich.

→ Hinweis: Auch in Pascal gibt es den Sprungbefehl GOTO. Er kann aber in allen Fällen vermieden und durch andere Strukturen ersetzt werden. Im Hinblick auf ein systematisches und übersichtliches Programmieren wird auf diese Möglichkeit in diesem Buch nicht eingegangen.

Regeln

→ Alle Pascal-Befehle werden durch ein Semikolon (;) voneinander getrennt.

→ Pascal-Befehle werden immer in der Reihenfolge abgearbeitet, wie sie auftreten.

→ Die Anweisungen des Programms werden zwischen BEGIN und END geschachtelt.

→ Nach BEGIN kein Zeichen!

→ Vor END kann ein Semikolon stehen, sollte jedoch nicht. (Nur ein Semikolon allein erzeugt nämlich eine leere Anweisung.)

→ Nach END ein Punkt, wenn es das Ende des Hauptprogramms darstellt, sonst ein Semikolon.

einfache Programm-Struktur

Ein einfaches Programm, das nur aus einem Hauptprogramm besteht, d.h. ein Programm ohne Unterprogramme (Prozeduren/Funktionen) hat demnach folgende Form:

```
PROGRAM programmname;

USES...;
CONST...;
TYPE...;
VAR...;

BEGIN
  ....
  ....
  Anweisungen (durch Semikolon getrennt)
  ....
  ....
END.
```

Ein Programm mit Prozeduren und Funktionen hat die Form:

Programm-Struktur mit Prozeduren und Funktionen

```
PROGRAM programmname;

  USES...;
  CONST...;
  TYPE...;
  VAR...;

  PROCEDURE prozedurname;
    CONST...;
    VAR...;
      BEGIN
      ...
      END;

  PROCEDURE prozedurname (variablendeklaration);
    CONST...;
    VAR...;
      BEGIN
      ...
      END;

  PROCEDURE prozedurname (VAR variablendeklaration);
    CONST...;
    VAR...;
      BEGIN
      ...
      END;

  PROCEDURE prozedurname (variablen, VAR variablen);
    CONST...;
    VAR...;
      BEGIN
      ...
      END;

  FUNCTION funktionsname (variablendeklaration) : typ;
    CONST...;
    VAR...;
      BEGIN
      ...
      END;

  BEGIN (* Hauptprogramm *)
    ...
  END.
```

Übungen:

3.1.1 Was gehört alles zum Deklarationsteil eines Programms?

3.1.2 Nehmen Sie Stellung zu der Aussage: Vor einem END darf ein Semikolon stehen, sollte aber nicht.

3.1.3 Schreiben Sie ein Pascal-Programm „Zähneputzen". Überlegen Sie sich, was im Deklarationsteil stehen muß, und achten Sie auf die Reihenfolge der Anweisungen.

3.1.4 Zeigen Sie Ähnlichkeiten zwischen Kochrezepten und Pascal-Programmen auf.

3.2 Ein- und Ausgabebefehle – ein kleines Programm

Nun wird es aber bezüglich der Programmierung in Pascal ernst. Wir wollen, nachdem einige einfache Befehle geklärt sind, ein weiteres kleines Programm schreiben.

Um mit dem Computer über ein Programm in Kontakt treten zu können, brauchen wir sogenannte Ein- und Ausgabebefehle (die Ein- und Ausgabebefehle beziehen sich vorerst nur auf Eingaben mit der Tastatur und Ausgaben auf den Bildschirm):

WRITE WRITELN (Ausgabe)

WRITE und WRITELN:
Diese Befehle sind Ausgabebefehle, die übersetzt soviel heißen wie „schreibe". WRITE wird immer dann gebraucht, wenn nach dem geschriebenen Wort oder der geschriebenen Zahl in der gleichen Zeile weitergeschrieben werden soll.

CR (Zeilenvorschub)

Wenn nach der Ausgabe ein Zeilenvorschub (Carriage Return oder kurz: CR) folgen soll, d.h., wenn das Nächstgeschriebene in einer neuen Zeile anfangen soll, so wird WRITELN (sprich writeline) verwendet.

Leerzeile

Steht WRITELN alleine, so wird nur eine Leerzeile geschrieben. Ansonsten werden die Befehle gefolgt von runden Klammern, in denen sich das zu Schreibende befindet.

Beispiele: WRITELN (Wert) schreibt den Dateninhalt von Wert.
WRITELN (Wert1,Wert2,Wert3) wie oben, aber für drei Daten.
WRITELN ('Dies ist ein Beispiel') schreibt den Satz: „Dies ist ein Beispiel" auf das Ausgabegerät (meistens Bildschirm).

Mehrere Daten

Wenn mehrere Daten, Variablen, Wörter, Sätze usw. ausgegeben werden sollen, so werden sie in der Klammer jeweils durch ein Komma voneinander getrennt.

Text

Wenn Text in der Klammer steht, so muß er stets durch Hochkomma angeführt und beendet werden. (Das Hochkomma wird natürlich später nicht mit ausgegeben!)

READ READLN (Eingabe)

READ und READLN:
Dies sind Eingabebefehle, die übersetzt soviel wie „lies" heißen. Mit READ wird ein Zeichen an den Rechner übergeben. Die Eingabe erfolgt durch Tastendruck (also ohne die Return-Taste).

READLN (sprich readline) ermöglicht es, eine Zahl oder ein Wort einzugeben. Der Name der einzugebenden Variablen steht wieder in einer Klammer hinter dem READLN-Befehl. Sollten mehrere Daten mit einem Befehl eingegeben werden, so sind sie in der Klammer durch Kommata zu trennen – hierbei aber Vorsicht: Mehrere einzugebende Daten in einem READLN-Befehl könnten später im Programmlauf seltsame Effekte ergeben.

Die READLN-Eingabe wird mit der Return-Taste abgeschlossen.

Beispiel: READLN (Wert) übergibt eine einzugebende Zahl der Variablen Wert (vorausgesetzt, der Name Wert steht für eine Zahlenvariable).

<Return>

Später im Programmlauf muß nach dem Eintippen der Zahl (oder des Wortes) die Return-Taste gedrückt werden, um dem Rechner zu signalisieren, daß die Eingabe beendet ist.

Ausgabe

Wir wollen nun ein kleines Programm schreiben, das einige Sätze auf unseren Bildschirm schreibt.

Jetzt aber . . .

Gehen Sie in den Editor!

Geben Sie nun nach dem I(nsert-Befehl folgenden Programmtext ein:

Beispiel

```
PROGRAM Saetze;

BEGIN
  WRITELN ('Dies ist ein kleines Programm,');
  WRITELN ('das ein paar Worte auf den Bildschirm');
  WRITELN ('bringen soll.')
END.
```

→ Beachten Sie: Vor dem END sollte kein Semikolon stehen, kann aber.

END

Sollten Sie keine Kleinbuchstaben zur Verfügung haben, schreiben Sie einfach groß. Andererseits dürfen bei den meisten Systemen auch die Befehlswörter klein geschrieben werden – in den folgenden Programmen werden Befehlswörter (reservierte Wörter, Kap. 4) vom anderen Text optisch getrennt.

Nachdem Sie dieses Programm im Editor eingetippt haben, lassen Sie den Editor den Text mit <etx> (meistens Ctrl-C) annehmen, und verlassen Sie den Editor. Dann rufen Sie den Compiler auf oder tippen einfach R für R(UN, damit das Programm übersetzt und gestartet wird.

Nun sollte der Rechner den folgenden Text auf dem Bildschirm schreiben:

```
Dies ist ein kleines Programm,
das ein paar Worte auf den Bildschirm
bringen soll.
```

Bildschirm löschen (PAGE)

Es könnte sein, daß dies nicht der einzige Text auf Ihrem Bildschirm ist, sondern sich noch die Kommandozeile oder irgendein früher geschriebener Text auf dem Bildschirm befindet.

Fügen Sie einfach nach BEGIN eine Zeile ein, die so aussieht:

```
PAGE (OUTPUT);
```

Dieser Befehl bewirkt, daß der Bildschirm gelöscht wird.

Bildschirm-Koordinaten (GOTOXY)

Wenn wir die Ausgabe an einer bestimmten Stelle des Bildschirms haben wollen, so verwenden wir

```
GOTOXY (x,y);
```

mit x,y ganzzahlige Koordinaten der entsprechenden Bildschirmstelle (die möglichen Werte für x und y hängen natürlich vom jeweiligen Bildschirm ab).

Ausgabe Formate

Ein Beispiel:

Beispiel für GOTOXY

```
PROGRAM Bildschirmtest;

BEGIN
  GOTOXY (10,20);
  WRITELN ('Noch ein... ');
  GOTOXY (15,30);
  WRITELN ('Testtext. ')
END.
```

Im folgenden noch einige Anmerkungen zu den Ausgabebefehlen, die Sie erst verwenden können, wenn Sie mit Zahlen und Variablen arbeiten:

Ausgabe mit Format Formatierte Ausgabe:

Ganze Zahlen Wenn ganze Zahlen mit einer bestimmten Stellenzahl ausgegeben werden sollen, so benutzt man die Form:

WRITELN (Ganzzahl:Stellenzahl)

Beispiel: WRITELN (Wert1:4, Wert 2:5) bewirkt, daß Wert1 mit vier Stellen, Wert2 mit fünf Stellen rechtsbündig ausgegeben wird.

Dezimalzahlen Sollen Dezimalzahlen mit einer bestimmten Stellenzahl und einer bestimmten Nachkommastellenzahl ausgegeben werden, so ist die Form:

WRITELN (Dezzahl:Stellenzahl:Nachkomma)

Beispiel: WRITELN (Wert:7:2) bewirkt, daß Wert mit insgesamt sieben Stellen und davon zwei Stellen hinter dem Dezimalpunkt ausgegeben wird.

→ Achtung: Der Dezimalpunkt zählt bei der Stellenzahl mit!

Beispiele *Noch einige Beispiele:*

Variable:	Inhalt:	Ausgabebefehl:	Ausgabe:
Wert1	123	WRITELN(Wert1:5)	→ 123
Wert2	12	WRITELN(Wert2:2)	→12
Wert3	12.4	WRITELN(Wert3:5:2)	→12.40
Wert4	12.4	WRITELN(Wert4:7:3)	→ 12.400

Sie sollten als guter Programmierer darauf achten, daß der Inhalt der Variablen die vorgegebene Stellenzahl nicht überschreitet, da die verschiedenen Rechner unterschiedlich darauf reagieren!

Übungen:

3.2.1 Was ist der Unterschied zwischen WRITE und WRITELN?

3.2.2 Welche Ausgabe bewirken folgende Programmzeilen?

Deklarationsteil

```
Zahl := 15.37;
WRITELN (Zahl:10:4);
```

3.2.3 Schreiben Sie ein Programm, das einen Satz einliest und wieder ausgibt. Erweitern Sie Ihr Programm so, daß der eingelesene Satz zehnmal nacheinander ausgegeben wird.

3.2.4 Schreiben Sie ein Programm, das ein Viereck, dessen Seitenlinien aus * bestehen, auf den Bildschirm zeichnet.

3.2.5 Was geschieht, wenn der Bildschirm voll ist?

3.2.6 Programme bestehen meistens aus folgenden Teilen:

Eingabe;
Verarbeitung;
Ausgabe.

Erläutern Sie diese Aussage!

3.3 Konstanten und Variablen

Beispiel

```
PROGRAM Beispiel;

  CONST Pi = 3.1416;

  VAR Radius, Flaeche : REAL;

  BEGIN
    WRITE ('Radius: ');
    READLN (Radius);
    Flaeche := Pi * Radius * Radius;
    WRITELN ('Flaeche=',Flaeche:7:2)
  END.
```

Das im letzten Abschnitt erstellte Programm bestand nur aus dem Programmkopf und dem Hauptprogramm zwischen BEGIN und END. Es hatte noch keine Konstanten und Variablen.

Variable

Variable:
Variablen sind Speicherplätze im Rechner, in denen Zahlen, Buchstaben, Worte oder andere Objekte (die wir später noch kennenlernen) gespeichert werden. Eine Variable hat einen Namen (Variablenname), einen sogenannten Datentyp und einen Dateninhalt. Mit der Variablen kann genauso gearbeitet werden wie mit den Daten selbst. Mit einer Zahlenvariablen kann z. B. genauso gerechnet werden wie mit anderen Zahlen. Der Name der Variablen und ihr Typ werden im Deklarationsteil des Programms festgelegt. Der Dateninhalt wird der Variablen aber erst im Programm zugewiesen und kann jederzeit geändert werden.

Konstante

Konstante:
Eine Konstante ist ähnlich der Variablen ein Speicherplatz, der einen Namen und einen Dateninhalt hat. Jedoch läßt sich der Dateninhalt der Konstanten nicht mehr im Programm verändern. Der Inhalt wird im Deklarationsteil festgelegt.

Deklarationsteil

→ *Merke:* Wir unterscheiden für Konstanten und Variablen:

Name/Typ/Inhalt

Name – Datentyp – Dateninhalt.

Bei Variablen werden Namen und Datentypen im Deklarationsteil festgelegt und die Dateninhalte im Programm zugewiesen.

Bei Konstanten werden Namen und Dateninhalte (und damit Datentypen) nur im Deklarationsteil festgelegt.

Global/Lokal

Global:
Konstanten und Variablen, die im Deklarationsteil des Programms festgelegt sind, heißen global. Sie können überall im Programm verwendet werden (also auch in den Prozeduren und Funktionen).

Später (Kap. 9/10) werden wir noch lokale Konstanten und Variablen kennenlernen, die nur in der jeweiligen Prozedur/Funktion gültig sind.

erlaubte Namen

Variablennamen:
Die Variablennamen sind beliebig und können in der Regel bis zu acht signifikante Zeichen enthalten. Das erste Zeichen muß ein Buchstabe sein. Sonderzeichen, die von Pascal verwendet werden, sowie Namen von Pascal-Befehlen dürfen nicht verwendet werden (reservierte Wörter, siehe Kap. 4).

Beispiele

Beispiele für richtige und falsche Variablen- und Konstantennamen:

```
A        : richtig
Test     : richtig
x3       : richtig
1a       : falsch (1. Zeichen muß Buchstabe sein)
X-oben   : falsch (kein Sonderzeichen)
```

Deklaration

CONST:
Die Deklaration der Konstanten beginnt mit dem Wort CONST. Danach werden, durch Semikolon getrennt, die Konstanten aufgeführt, indem den Konstantennamen durch ein Gleichheitszeichen ein Wert zugewiesen wird.

VAR:
Die Variablendeklaration beginnt mit dem Wort VAR und zählt danach die Namen der Variablen und ihren Typ auf. Die Form: Variablenname, Doppelpunkt, Typ. Verschiedene Variablen gleichen Typs können vor dem Doppelpunkt, durch Kommata getrennt, aufgezählt werden. Mehrere Variablendeklarationen werden durch Semikolon voneinander getrennt.

Beispiel

Beispiel:

```
CONST Vier=4;
      Dez=5.7;
      Elf=11;
      Neun=10;
```

Einfache Datentypen

```
VAR    Wert : INTEGER;
       Zahl,Volumen : REAL;
```

Damit wir mit der Variablendeklaration etwas anfangen können, werden hier kurz die sogenannten einfachen Datentypen dargestellt. (Genaue Beschreibung siehe Kap. 5.)

Datentypen

Einfache Datentypen:

INTEGER:
Ganze Zahl. Bei jedem Rechner ist darauf zu achten, daß der Zahlenbereich beschränkt sein kann. Bei sogenannten 8-Bit-Rechnern können ganze Zahlen zwischen −32767 und +32768 liegen. Für größere ganze Zahlen gibt es oft den Typ: LONG INTEGER (UCSD-Pascal).

REAL:
Dezimalzahl. Eine Dezimalzahl kann auf einem Computer ebenfalls nur mit einer beschränkten Genauigkeit angegeben werden. Sollten bei sehr kleinen Zahlen die Stellen hinter dem Dezimalpunkt oder bei großen Zahlen die Stellen davor nicht ausreichen, so werden die Zahlen in der Zehnerpotenz-Schreibweise (wie beim Taschenrechner) angegeben.

Beispiel: Statt 300000000000 kann 3E11 (sprich: 3 mal 10 hoch 11) angegeben werden.

CHAR:
Zeichen. Eine Variable dieses Typs kann ein beliebiges Zeichen aus dem Zeichensatz des Rechners enthalten.

STRING:
Zeichenkette. Eine Zeichenkette ist eine Folge von beliebigen Zeichen. Ein String kann z. B. ein Wort, ein Name oder ein Satz sein.

BOOLEAN:
Logische Variable. Dieser Variablentyp kennt nur zwei mögliche Werte: Wahr oder Falsch. In Pascal heißen diese Werte:

TRUE – Wahr
FALSE – Falsch

Im Programm kann also einer Booleschen Variablen der Wert TRUE oder FALSE zugewiesen oder dieser Wert abgefragt werden, aber Vorsicht:

→ Eine Variable vom Typ BOOLEAN wird in UCSD-Pascal nicht ausgedruckt!

Nun noch ein kleines Beispielprogramm zur Übung des Umgangs mit Konstanten und Variablen.

Einrücken

Damit der Programmtext etwas übersichtlicher wird, rückt man für gewöhnlich einige Textzeilen ein. Außerdem sollte man möglichst einige Kommentare in den Text einfügen.

Kommentar

Kommentar läßt sich in geschweiften Klammern einfügen. Er wird beim Übersetzen des Programms vom Rechner nicht berücksichtigt. Wenn keine geschweiften Klammern vorhanden sind, so können Ersatzzeichen verwendet werden:

(* Kommentar *)

Programm

```
PROGRAM Vartest;
 (* Ein Programm zur Demonstration einfacher Typen *)

  CONST Drei=3;
        Elfkommadrei=11.3;

  VAR   Ganz1,Ganz2 : INTEGER;
        Name : STRING;

BEGIN
  PAGE(OUTPUT);
  (* Bildschirm löschen *)

  (* Eingabe von Zahlen und Worten: *)
  WRITELN('Dies ist ein Testprogramm:');
  WRITELN;
  WRITE('Geben Sie eine ganze Zahl ein: ');
  READLN(Ganz1);
  WRITELN;
  WRITE('Noch eine: ');
  READLN(Ganz2);
  WRITE('Geben Sie Ihren Namen ein: ');
  READLN(Name);
  (* Ende der Eingabe *)

  (* Ausgabeteil: *)
  PAGE(OUTPUT);

  WRITELN('Lieber ',Name,' !');
  WRITELN;
  WRITELN('Die Konstanten dieses Programms');
  WRITELN('sind ',Drei,' und ',Elfkommadrei);
  WRITELN;
  WRITELN('Sie haben folgende Zahlen eingegeben:');
  WRITELN(Ganz1,' und ',Ganz2);
  WRITELN;
  WRITELN('...das war es.')

END. (* Programmende *)
```

Programmtest

Tippen Sie dieses Programm bitte im Editor ein, und bringen Sie es mit R(UN zum Laufen.

Experimentieren Sie mit verschiedenen Eingaben, und ändern Sie das Programm nach Ihren Wünschen noch ein wenig.

Übungen:

3.3.1 Welcher Fehler steckt im folgenden Programm?

```
PROGRAM Test;
   CONST Elf=11;
   BEGIN
      Elf := Elf * 2;
      WRITELN (Elf)
   END.
```

Druckerausgabe

3.3.2 Durch welche Anweisung läßt sich der Dateninhalt der Variablen WERT um die Zahl 4 erhöhen? (WERT sei vom Typ INTEGER.)

3.3.3 Wie unterscheiden sich CHAR und STRING?

3.3.4 Was geschieht, wenn Sie in das Programm Vartest als Zahlen 3.5 und −2.8 eingeben? Korrigieren Sie das Programm so, daß auch solche Eingaben möglich sind.

3.3.5 Was geschieht, wenn Sie statt Ihres Namens eine Zahl eingeben?

3.4 Wie hilft sich der stolze Besitzer eines Druckers in UCSD-Pascal?

Diese Frage stellt den Anfänger und oft sogar den Fortgeschrittenen vor schier unüberwindbare Probleme. Der Grund liegt darin, daß zur Verwendung des Druckers an sich schon höhere Datentypen und der Umgang mit sogenannten Dateien vorausgesetzt werden. Wir wollen hier noch nicht auf die exakten Hintergründe eingehen und nur ein „Kochrezept" angeben, wie man den Drucker vom Programm aus verwendet.

„Kochrezept"

Im Band 2 wird der verwendete Datentyp näher erläutert.

Zuerst denken Sie sich einen Namen für den Drucker aus, denn er wird vom Programm so ähnlich behandelt wie eine Variable.

INTERACTIVE

Nehmen wir an, wir nennen ihn „Drucker". Im Deklarationsteil des Programms erscheint der Drucker nur als Variable vom Typ INTERACTIVE.

```
VAR Drucker : INTERACTIVE;
```

Bevor der Drucker erstmalig angesprochen wird, sollte im Programm der Befehl: REWRITE (Drucker, 'PRINTER:') oder REWRITE (Drucker, '#6:') stehen. Dieser Befehl bewirkt, daß eine Datei namens Drucker über den Datenkanal PRINTER: (oder, was das gleiche bedeutet ,#6:) eröffnet wird.

Datenkanal eröffnen

Bei jedem WRITE- oder WRITELN-Befehl im Programm, der eine Ausgabe auf den Drucker bewirken soll, muß als erste Variable in der Klammer der Variablenname des Druckers erscheinen.

Ausgabe

Zum Schluß des Programms, wenn der Drucker nicht mehr gebraucht wird, sollte der Datenkanal mit CLOSE (Drucker) wieder geschlossen werden.

Datenkanal schließen

Ein kleines Beispielprogramm:

Programm

```
PROGRAM Drucktest;

  VAR Satz : STRING;
      Drucker : INTERACTIVE;
```

Listing drucken

(Fortsetzung des Programms)

```
BEGIN
  PAGE(OUTPUT);
  WRITELN('Dies ist ein Druckertest: ');
  WRITELN('Geben Sie eine Textzeile ein: ');
  READLN(Satz);

  (* Eroeffnen des Druckers: *)
  REWRITE (Drucker,'PRINTER:');

  (* Schreiben auf dem Bildschirm: *)
  WRITELN(Satz);

  (* Schreiben auf dem Drucker: *)
  WRITELN(Drucker,Satz);
  WRITELN(Drucker,'Und noch einmal: ');
  WRITELN(Drucker,Satz);

  (* Schliessen des Druckerkanals: *)
  CLOSE(Drucker)
END. (* Programmende *)
```

Listing Programmlistings:

Zum Auflisten des Programmtextes rufen wir den Filer auf und wählen T für T(ransfer.

Nun wird das entsprechende Textfile auf den Drucker (PRINTER: oder #6:) übertragen.

Beispiele:

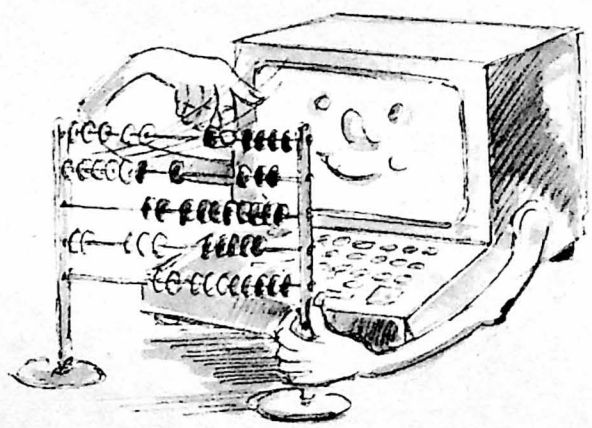
```
Transfer what file? #4:SYSTEM.WRK.TEXT,#6:
Transfer what file? #5:MEINPROG.TEXT,#6:
```

Übungen:

3.4.1 Erstellen Sie Programmlistings einiger Programme. Wenn Sie noch keine Programme selber erstellt haben, so benutzen Sie Textfiles von APPLE3:.

3.4.2 Schreiben Sie das Programm Vartest aus Kap. 3.3 so um, daß die Ausgaben auf einen Drucker gehen.

3.5 Pascal kann auch rechnen – Beispiele



Wertzuweisung (:=)

Der Name unserer Maschine – Computer oder Rechner – hat damit zu tun, daß sie rechnen kann. Dies ist natürlich richtig. Man sollte an dieser Stelle allerdings anmerken, daß die Rechengenauigkeit und Rechengeschwindigkeit preiswerter Tischcomputer nicht überwältigend ist und von vielen Taschenrechnern oft übertroffen wird.

Rechengenauigkeit

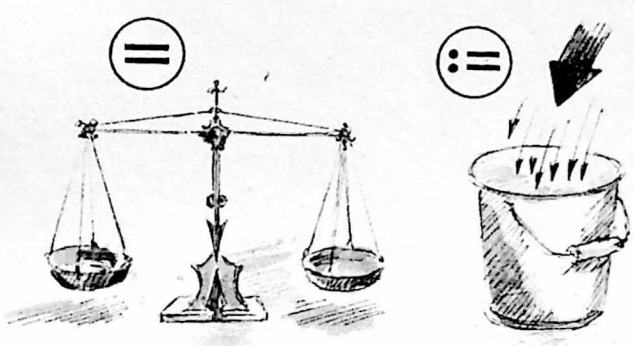
Die Stärke des Tischcomputers liegt aber auch gar nicht auf diesem Gebiet, sondern vielmehr darin, daß er große Datenmengen sehr schnell verarbeiten kann und in der Lage ist, Texte zu verarbeiten.

Trotzdem kann unser Tischrechner natürlich rechnen. Auch die Rechengenauigkeit reicht für die meisten Anwendungen allemal aus!

Bevor wir uns den in Pascal möglichen arithmetischen Operationen zuwenden, sollten wir einige prinzipielle Überlegungen zum Rechnen anstellen.

→ Wertzuweisung: Wenn man einer Variablen einen Wert zuweisen möchte (z.B. eine Zahl oder das Ergebnis einer Rechenoperation zuordnen will), so benutzt man stets einen Doppelpunkt, der von einem Gleichheitszeichen gefolgt wird: := (gesetzt gleich).

Wertzuweisung/ Gleichheit



Wir müssen sehr genau zwischen = und := unterscheiden.

= oder :=

Das einfache Gleichheitszeichen bedeutet, daß die beiden Ausdrücke rechts und links des Zeichens wirklich gleich sind oder es wird in Aussagen gebraucht, die wahr oder falsch sein können. Bei der Deklaration von Konstanten z.B. ist dies der Fall. Auch wenn wir später sogenannte Entscheidungsbefehle kennenlernen, werden wir das einfache Gleichheitszeichen benutzen. Da wird dann nämlich gefragt, ob ein Ausdruck *gleich* einem anderen ist!

=

Anders sieht es bei dem := Zeichen aus. Hierbei handelt es sich um eine sogenannte Zuweisungsoperation. Wir könnten es auch verstehen als ←. Dem Ausdruck vor diesem Zeichen wird der Ausdruck nach dem Zeichen zugeordnet. Logischerweise sollte man dieses Zeichen von rechts nach links lesen (bzw. verstehen).

:=

Beispiele:

Beispiele

Die Variable Wert sei vom Typ INTEGER, Dez vom Typ REAL.

Rechenoperationen

Wert:=3; bedeutet, daß der Variablen Wert die ganze Zahl 3 zugeordnet wird. Mit der Variablen Wert kann im folgenden genauso gerechnet werden wie mit der Zahl 3.

Der Befehl Dez:=Wert/2; ordnet der Variablen Dez die Dezimalzahl 1.5, nämlich Wert dividiert durch 2 zu.

„Speichertrick"

Eine für Anfänger häufig etwas undurchsichtige Operation sieht folgendermaßen aus:

Wert:=Wert + Wert;

Diese Operation berechnet Wert + Wert, d.h. 3 + 3, und ordnet es der Variablen Wert erneut zu, so daß die Variable Wert nach diesem Befehl den Zahlenwert 6 enthält.

Grundrechenarten

Nun aber zu den arithmetischen Operationen:

Grundrechenarten:

+ : Addition
− : Subtraktion
* : Multiplikation
/ : Division (mit Ergebnis vom Typ REAL)
DIV : Division für ganze Zahlen (Ergebnis vom Typ INTEGER)

Spezielles zur Division

Bemerkungen: Dividiert man zwei Zahlen mit Hilfe von /, so ist das Ergebnis stets vom Typ REAL (gleichgültig, von welchem Typ die Zahlen sind). Will man zwei ganze Zahlen (d.h. vom Typ INTEGER) durcheinander dividieren, und als Ergebnis wieder eine ganze Zahl erhalten, so benutzt man DIV als Rechenbefehl, und das Ergebnis ist vom Typ INTEGER. Sollte das Ergebnis der Division eine Dezimalzahl werden, so wird bei Benutzung von DIV als Ergebnis nur der Vorkommaanteil der Zahl benutzt (d.h. die Stellen nach dem Komma werden abgeschnitten, nicht gerundet!).

Die anderen Rechenoperationen haben als Ergebnis nur dann eine Zahl vom Typ INTEGER, wenn alle beteiligten Zahlen auch INTEGER sind.

Andere Operationen

Andere Rechenoperationen:

Im folgenden sind i und j vom Typ INTEGER, r ist REAL und x ist beliebig:

i MOD j : Modulo. Berechnet den ganzzahligen Rest bei der Division der ganzen Zahlen i und j (i DIV j).

ABS(x) : Absolutwert. Wandelt die Zahl x stets in eine positive Zahl.

TRUNC(r) : Berechnet den ganzzahligen Anteil der Dezimalzahl r. Das Ergebnis ist ganzzahlig. (Der Nachkommateil wird abgeschnitten, nicht gerundet!)

Berechnungen für ein Paket

ROUND(r) : Rundet die Dezimalzahl r zu einer ganzen Zahl.
SQR(x) : Quadrat von x.
PWROFTEN(x) : Zehn hoch x.
SQRT(x) : Quadratwurzel von x. (*)
SIN(x) : Sinus von x. (*)
COS(x) : Kosinus von x. (*)
ATAN(x) : Arcustangens von x (in Standard-Pascal: ARCTAN(x)). (*)
EXP(x) : Exponentialfunktion (e hoch x). (*)
LOG(x) : Logarithmus von x zur Basis 10(*)
LN(x) : Natürlicher Logarithmus von x. (*)

Die mit (*) gekennzeichneten Funktionen setzen als zweite Pascal-Anweisung 'USES TRANSCEND;' voraus.

→ Hinweis: Eine Standardfunktion zum Potenzieren ist in Pascal nicht vorgesehen. Wir können uns jedoch helfen mit:

$a^b = EXP(b * LN(a))$

Dies war eine kleine Übersicht über die Rechenbefehle in Pascal. Die meisten Befehle werden Sie vorerst wieder vergessen, sollten aber, wenn sie gebraucht werden wissen, wo man sie findet.

Kleine Beispielprogramme

1. Beispiel:

Beispiel PAKET

Aufgabe

In einer Firma hat ein Angestellter folgende Aufgabe: Er soll Pakete einpacken. Die Pakete sind von unterschiedlicher Größe. Der Angestellte will wissen, wieviel Platz im Lager benötigt wird (Volumen), wieviel Packpapier (Oberfläche) und wieviel Paketschnur er braucht.

Damit er nicht bei jedem Paket umständlich rechnen muß, schreiben wir dem Angestellten ein kleines Programm, in das er Länge, Breite und Höhe des Pakets eingibt und das alle anderen Werte berechnet.

Die Formeln: Sei Länge l , Breite b , Höhe h

Volumen: V = l * b * h
Oberfläche: O = 2 * (l*b + l*h + b*h)
Schnurlänge: L = 2*l + 2*b + 4*h + Schleife

Die Schnurlänge für die Schleife sei 10 cm.

Alle Längen werden in cm angegeben.

Paketprogramm

Wir planen erst einmal das Programm:

Blockdiagramm

Programm PAKET
Konstante Schleife=10 Variablen Länge,Breite,Höhe,Volumen, Oberfläche,Schnur : REAL
Anfang
Eingabe der Werte Länge,Breite,Höhe
Berechnung von Volumen,Oberfläche,Schnurlänge
Ausgabe der Werte
Ende.

Programm

```
PROGRAM Paket;

  CONST Schleife=10;   (* fuer die Schleife *)

  VAR Laenge, Hoehe, Breite, Volumen, Oberflaeche, Schnur : REAL;

  BEGIN (* Hauptprogramm *)
    PAGE (OUTPUT);            (* loescht Bildschirm *)
    (* Eingabeteil *)
    WRITELN ('Paketprogramm:');
    WRITELN ('  alle Werte in cm ');
    WRITELN;                  (* Leerzeile *)
    WRITE ('Geben Sie die Laenge ein: ');
    READLN (Laenge);
    WRITE ('Geben Sie die Breite ein: ');
    READLN (Breite);
    WRITE ('Geben Sie die Hoehe ein:  ');
    READLN (Hoehe);

    (* Berechnung *)
    Volumen := Laenge * Breite * Hoehe;
    Oberflaeche := 2*(Laenge*Breite + Laenge*Hoehe + Breite*Hoehe);
    Schnur := 2*Laenge + 2*Breite + 4*Hoehe + Schleife;

    (* Ausgabeteil *)
    WRITELN;
    WRITELN ('Das Volumen betraegt ',Volumen:6:2,' cm^3');
    WRITELN ('Sie brauchen ',Oberflaeche:6:2,' cm^2 Papier');
    WRITELN ('Die Schnur muss ',Schnur:5:2,' cm lang sein')
  END. (* Programmende *)
```

Ergebnis Ausgabe des Programms:

```
Paketprogramm:
  alle Werte in cm

Geben Sie die Laenge ein: 10.00
Geben Sie die Breite ein:  20.50
Geben Sie die Hoehe ein:  35.25

Das Volumen betraegt 7226.25 cm^3
Sie brauchen 2560.25 cm^2 Papier
Die Schnur muss 212.00 cm lang sein
```

Winkeltabelle

Frage: Hätten wir als Datentypen statt REAL auch INTEGER verwenden können?

2. Beispiel:

Beispiel WINKEL-TABELLE

Aufgabe

Im nächsten Programm wollen wir uns eine Tabelle der Sinus-, Kosinus- und Tangenswerte eines bestimmten Winkelbereichs ausgeben lassen. Dazu geben wir den Anfangs- und Endwinkel sowie die Schrittweite ein. Um den Umgang mit verschiedenen Datentypen zu üben, vereinbaren wir, daß es sich bei allen Winkeln nur um ganzzahlige Werte handeln soll. Da Winkelfunktionen in Bogenmaß rechnen, müssen wir die Winkel durch Multiplikation mit (π/180) umrechnen.

REPEAT-Schleife

Für dieses Problem brauchen wir einen Algorithmus, der erst später (Kap. 6.2) genauer beschrieben wird: eine Schleife. Die Konstruktion

```
REPEAT
   <Anweisungen>
UNTIL <Endbedingung>;
```

nennen wir Wiederhole...bis...-Schleife.

Blockdiagramm

Programm WINKELTABELLE
Konstante pi=3.14159 Variablen Winkel, Anfang, Ende, Schrittweite : ganze Zahlen x, sinus, kosinus, tangens : Dezimalzahlen
Anfang
Eingabe: Anfang, Ende, Schrittweite
Anfangswert für Winkel
Wiederhole
x:=Winkel * pi / 180
Berechne sinus, kosinus, tangens von x
Ausgabe: Winkel, sinus, kosinus, tangens
Erhöhe Winkel um Schrittweite
bis Winkel größer als Ende
Ende.

Das Programm:

Programm

```
PROGRAM Winkeltabelle;

USES TRANSCEND; (* fuer Winkelfunktionen *)

CONST pi = 3.14159;

VAR Winkel, Anfang, Ende, Schrittweite : INTEGER;
    x, sinus, kosinus, tangens : REAL;
```

Typenkonflikte

(Fortsetzung des Programms)

```
BEGIN
  WRITELN ('Erstellung einer Winkeltabelle: ');
  WRITELN;
  WRITE ('Eingabe Anfangswinkel: ');
  READLN (Anfang);
  WRITE ('Eingabe Endwinkel: ');
  READLN (Ende);
  WRITE ('Eingabe Schrittweite: ');
  READLN (Schrittweite);
  WRITELN;
  Winkel := Anfang;
  WRITELN (' Winkel:     sinus:          kosinus:          tangens: ');
  REPEAT
    x := Winkel * pi / 180;
    sinus := SIN (x);
    kosinus := COS (x);
    tangens := sinus / kosinus;
    WRITELN (Winkel:4, sinus:16:3, kosinus:16:3, tangens:16:3);
    Winkel := Winkel + Schrittweite
  UNTIL Winkel > Ende
END.
```

Aufgabe: Testen Sie das Programm auf einem Rechner!

Typenkonflikte:

Typenkonflikte REAL – INTEGER

Achten Sie unbedingt darauf, daß z. B. einer INTEGER-Variablen kein REAL-Wert zugeordnet werden kann.

Auch die Operationen DIV und MOD funktionieren natürlich nur mit INTEGER-Variablen.

Beispiel:

```
VAR A,B:REAL;
    C,D:INTEGER;

dann ist: C:=A              falsch
          C:=TRUNC(A)       richtig
          C:=A DIV B        falsch
          C:=C DIV D        richtig
          A:=C              richtig
```

Wenn Typenkonflikte vorkommen, so merkt der Compiler dies beim Übersetzen und gibt die Fehlermeldung: Type conflict in operands aus. Achten Sie schon bei der Planung des Programms darauf, welche Variablen miteinander verknüpft werden.

Übungen:

3.5.1 Worin liegt der Unterschied zwischen = und :=?

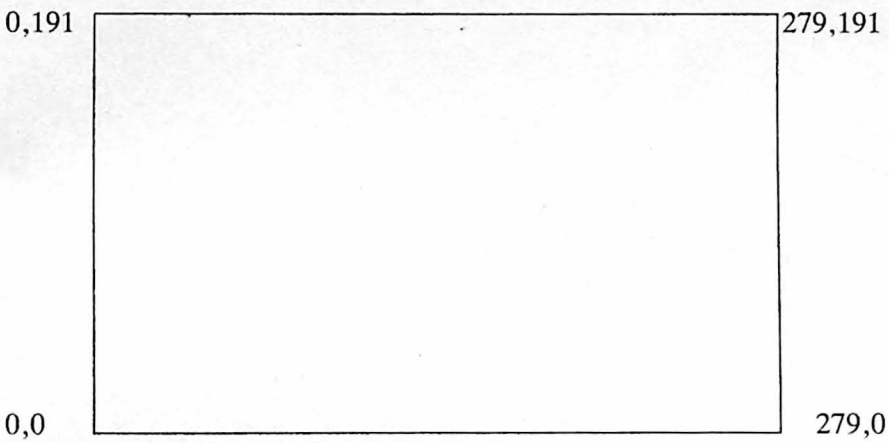
3.5.2 Welcher Unterschied besteht zwischen / und DIV?

3.5.3 Das folgende Programm enthält einige Fehler. Finden, berichtigen und begründen Sie diese:

```
PROGRAM Mit_Fehlern;

  VAR Liter, Preis, Verbrauch = REAL;
      Strecke, Literpreis = INTEGER;

  BEGIN;
    WRITE ('Geben Sie die Strecke in vollen km ein:');
    READLN (Strecke);
    WRITE ('Geben Sie den verbrauchten Sprit in Litern ein:');
    READLN (Liter);
    WRITE ('Geben Sie den Preis in DM ein:');
    READLN (Preis);
    Verbrauch = Preis * 100 DIV Strecke;
    Literpreis = Preis / Liter;
    WRITELN ('Sie haben',Verbrauch:4:2,' Liter/100km verbraucht');
    WRITELN ('Der Spritpreis: ',Literpreis:4:2,' DM/Liter');
  END.
```

3.5.4 Schreiben Sie ein Programm, das so lange eingegebene Dezimalzahlen addiert, bis eine Null eingegeben wird. Die Summe der Zahlen soll mit vier Stellen vor und zwei Stellen hinter dem Komma ausgegeben werden.

3.6 Mit Grafik geht manches besser



Koordinaten der Bildschirmecken

Um Grafiken zu erstellen, muß gleich nach dem Programmnamen die Anweisung USES TURTLEGRAPHICS folgen. Der Grafikbildschirm des APPLE wird dann mit dem Befehl INITTURTLE aktiviert und gelöscht. Durch PENCOLOR (Farbe) können wir die Farben BLACK und WHITE wählen. MOVETO (x,y) bewegt den Zeichenstift zu einer Bildschirmkoordinate x,y und hinterläßt einen Strich mit der gewählten Farbe. Die Bildschirmkoordinaten sind wählbar zwischen 0,0 (links unten) und 279,191 (rechts oben).

USES TURTLE-GRAPHICS

Wir wollen nun ein Rechteck zeichnen, dessen untere linke und obere rechte Ecke vom Benutzer eingegeben werden.

Aufgabe

Zeichnen eines Rechtecks

Blockdiagramm

Programm RECHTECK
Variablen Xlinks, Xrechts, Yunten, Yoben : ganze Zahlen
Anfang
Eingabe der Koordinaten
Bildschirm aktivieren
Farbe Schwarz wählen
Bewegung nach links unten
Farbe Weiß wählen
Bewegung nach rechts unten
Bewegung nach rechts oben
Bewegung nach links oben
Bewegung nach links unten
Ende.

Das Programm:

Programm

```
PROGRAM Rechteck;
  USES TURTLEGRAPHICS;

  VAR Xlinks, Xrechts, Yunten, Yoben : INTEGER;

  BEGIN
    WRITELN ('Koordinaten der unteren linken Ecke: ');
    WRITE ('X= '); READLN (Xlinks);
    WRITE ('Y= '); READLN (Yunten);
    WRITELN ('Koordinaten der oberen rechten Ecke: ');
    WRITE ('X= '); READLN (Xrechts);
    WRITE ('Y= '); READLN (Yoben);
    INITTURTLE;    (* Grafik aktivieren *)
    PENCOLOR (BLACK);    (*  Farbe  setzen *)
    MOVETO (Xlinks,Yunten);
    PENCOLOR (WHITE);    (* Zeichenfarbe setzen *)
    MOVETO (Xrechts,Yunten);
    MOVETO (Xrechts,Yoben);
    MOVETO (Xlinks,Yoben);
    MOVETO (Xlinks,Yunten)
  END.
```

Aufgabe: Testen Sie das Programm!

Verzögerung

Was fällt Ihnen auf?

Das Rechteck wird zwar gezeichnet, verschwindet aber blitzschnell wieder, wenn das Programm beendet ist.

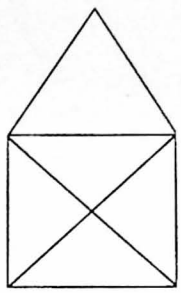
Abhilfe: Fügen Sie als letzte Anweisung READLN in das Programm ein. Diese Anweisung hat den Effekt, daß eine „leere" Eingabe mit <Return> abgeschlossen werden soll. Erst wenn die <Return>-Taste gedrückt ist, fährt das Programm fort.

Aufgabe: Denken Sie sich weitere Grafikprogramme aus.

Hinweis: Weitere Grafikbefehle sind im Kap. 8 genauer beschrieben.

Übungen:

3.6.1 Schreiben Sie ein Programm zum Zeichnen eines „Hexenhauses".



3.6.2 Ein Programm soll fünf Koordinaten abfragen und diese miteinander als Grafik verbinden.

3.7 Zusammenfassung

Im Kapitel 3 haben wir die Struktur von Pascal-Programmen kennen- und anwenden gelernt. Wir kennen den formalen Aufbau eines Programms und wissen, was Konstanten und Variablen sind.

Das Wissen über einfache Ein- und Ausgabebefehle, verbunden mit einer Übersicht der einfachen Datentypen und der Rechenoperationen in Pascal, versetzt uns in die Lage, schon recht erfolgreich Programme (insbesondere Rechenprogramme) zu erstellen.

Wir kennen die Tücken von Typenkonflikten und werden sie (hoffentlich) in Zukunft vermeiden.

Eine Hilfe dazu sind die Übersichtstabellen zu Rechenoperationen und Standardfunktionen in Verbindung mit den zugehörigen möglichen Datentypen am Anhang.

Reservierte Wörter und Symbole

Kapitel 4

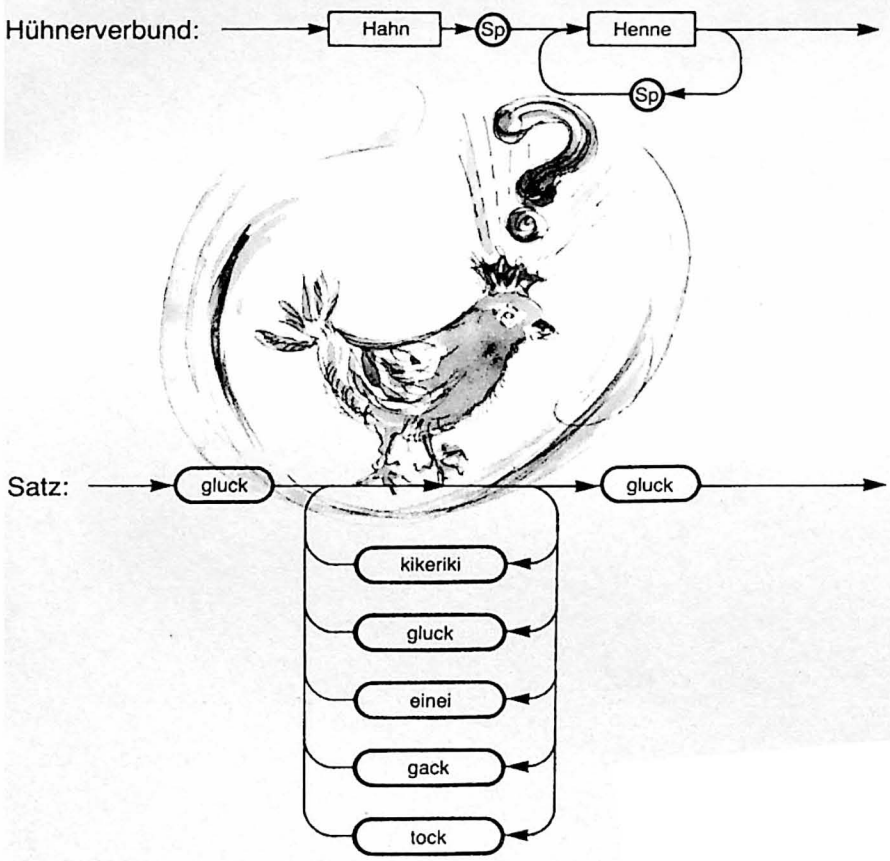
In diesem Kapitel wird kurz auf die in Pascal reservierten Wörter und die zur Programmierung benutzbaren Symbole eingegangen.

Reservierte Wörter sind Wörter, die vom Benutzer nicht verwendet oder umbenannt werden dürfen, da sie z. B. dem Befehlssatz der Programmiersprache entstammen.

Außerdem wird, unterstützt von einigen Beispielen, gezeigt, wie man Syntaxdiagramme liest, die vorschreiben, nach welcher Grammatik die Programmiersprache Pascal zu gebrauchen ist.

Nach dem Lesen dieses Kapitels

- wissen wir, was reservierte Wörter sind;
- kennen wir die reservierten Wörter und benutzbaren Symbole;
- wissen wir, was Syntax heißt und was ein Syntaxdiagramm ist;
- können wir Syntaxdiagramme lesen.



Liste reservierter Wörter

4.1 Zusammenstellung der Wörter und Symbole

Zeichensatz für Bezeichner

In der Namensgebung für unsere Programme, Variablen, Prozeduren und Funktionen unterliegen wir einigen Beschränkungen und Vorschriften. Wir dürfen unter Verwendung der folgenden Zeichen:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
(und in UCSD–Pascal: _)

Namen nach bestimmten Regeln, die wir in Kapitel 4.2 lernen, bilden.

Wir nennen diese Namen: benutzerdefinierte Bezeichner.

Reservierte Wörter

Diese dürfen jedoch nicht identisch sein mit sogenannten reservierten Wörtern.

Reservierte Wörter: Dies sind Wörter, die in der Programmiersprache Pascal bestimmte Bedeutungen haben und die dem Compiler eine korrekte Übersetzung und Ausführung des Programms ermöglichen.

Selbstverständlich darf z. B. eine Variable nicht denselben Namen haben wie eine Pascal-Anweisung, da es sonst zu unlösbaren Konflikten käme.

Folgende reservierte Wörter sind vorhanden:

Reserviertes Wort:	(deutsche) Bedeutung:
PROGRAM	Programm
INPUT	Eingabe
OUTPUT	Ausgabe
BEGIN...END	Anfang...Ende
USES	Benutzt
CONST	Konstante(n)
VAR	Variable(n)
TYPE	Typ
SEGMENT	Segment
PROCEDURE	Prozedur
FUNCTION	Funktion
FORWARD	Vorwärts
INTEGER	Ganze Zahl
REAL	Dezimalzahl
CHAR	Zeichen
STRING	Zeichenkette
BOOLEAN	Wahrheitswert
ARRAY...OF...	Feld...von...
PACKED ARRAY...OF...	Komprimiertes Feld...von...
SET...OF...	Menge...von...
RECORD...END	Verbund...Ende
FILE	Datei

Liste der Symbole

TEXT	Textdatei
INTERACTIVE	Interaktiv
EXTERNAL	Außerhalb
IMPLEMENTATION	Implementierung
RESET	Dateieröffnung
REWRITE	Dateieröffnung
CLOSE	Dateiabschluß
SEEK	Suche
PUT	Dateiausgabe
GET	Dateieingabe
NEW	Neue Zeigervariable
NIL	Anfang einer Zeigerliste
READ	Lies
READLN	Lies Zeile
WRITE	Schreibe
WRITELN	Schreibe Zeile
EXIT	Verlasse
REPEAT...UNTIL...	Wiederhole...bis...
WHILE...DO...	Solange...tue...
FOR...TO...DO...	Zähle...herauf bis...und tue...
FOR...DOWNTO...DO...	Zähle...herunter bis...und tue...
WITH...DO...	Mit...tue...
IF...THEN...ELSE	Wenn...dann...sonst
CASE...OF...END	Falls...von...Ende
AND	Und
OR	Oder
NOT	Nicht
MAXINT	Max. ganze Zahl
TRUE	Wahr
FALSE	Falsch
EOLN	Zeilenende
EOF	Dateiende
DIV, MOD	Rechen- und Ordnungs-
SQR, SQRT, ABS, TRUNC	operationen
ROUND	.
SIN, COS, ATAN (oder ARCTAN)	.
EXP, LN	
SUCC, PRED, ORD, CHR, ODD	
COPY, LENGTH, INSERT	
DELETE, STR	String-Operationen

Weiterhin verwendet Pascal folgende Symbole:

Symbole

( )	
[ ]	Ersatzsymbol: (. .)
{ }	Ersatzsymbol: (* *)

```
. , ; :
+ - * /
= <> < <= > >=
:=
^
```

→ *Merke:* Benutzerdefinierte Bezeichner (Namen) dürfen niemals reservierten Wörtern gleichen!

4.2 Syntaxdiagramme, und wie man sie liest

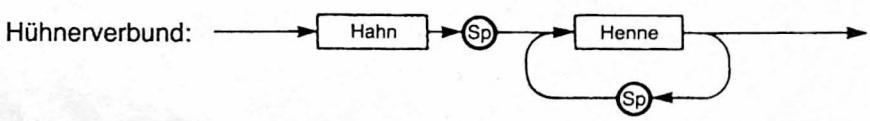
Syntax

Jeder Konstruktion in Pascal, sei es eine bestimmte Anweisung, eine zusammengehörige Folge von Anweisungen (Block) oder ein Bezeichner, liegen Regeln zugrunde, wie sie zu bilden ist. Diese Grammatik (Syntax) ist genau festgelegt, so daß es niemals Mißverständnisse oder undefinierte Situationen zwischen Programmierer und Computer geben kann. Alles ist eindeutig festgelegt.

Aus der „Hühnersprache"

So sollen, wie aus stets gut unterrichteten Kreisen bekannt wurde, einige Hühner dank eines neuartigen Kraftfutters zu solcher Intelligenz gekommen sein, daß sie sich einen Verwaltungscomputer zugelegt haben. Dieser Rechner ist in der Sprache DUCK zu programmieren.

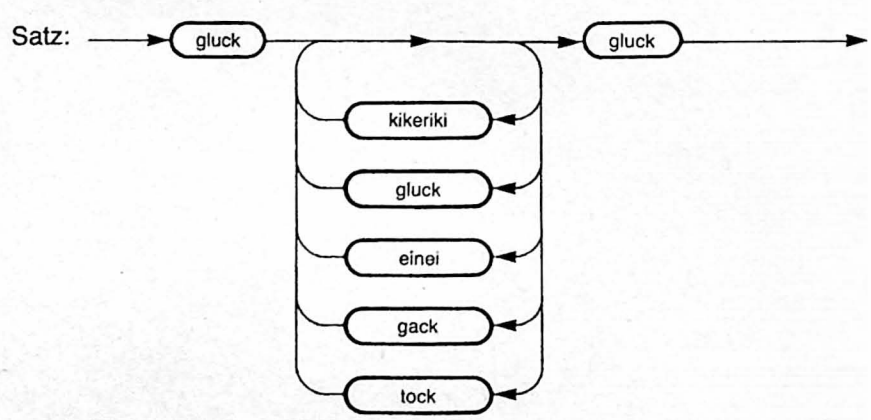
Diese Sprache kennt z. B. folgende Verbindungen:



Sp steht für Sprosse auf der Hühnerleiter.

Dies bedeutet: Der kleinste Verbund besteht aus einem Hahn und einer Henne, getrennt durch eine Sprosse. Die Schleife zeigt an, daß ein Hahn auch mehrere Hennen (getrennt durch Sprossen) haben kann. Nach diesem Syntaxdiagramm für „Hühnerverbund" kann jedoch eine Henne nie mehrere Hähne haben. Außerdem steht der Hahn stets an erster Stelle.

Auch die Sprache ist geregelt. Ein Satz in der Hühnersprache wird erstellt nach folgendem Syntaxdiagramm:



Ziffer/Buchstabe

So sind folgende Sätze syntaktisch korrekt:

```
gluck gluck
gluck gluck gluck
gluck einei gluck
gluck gack gack gluck
gluck gack tock gack gluck
```

Jedoch reagiert der Computer bei den Sätzen

```
gluck
kikeriki gluck gluck
gluck gack tock
```

mit einem „stupid-chicken-error".

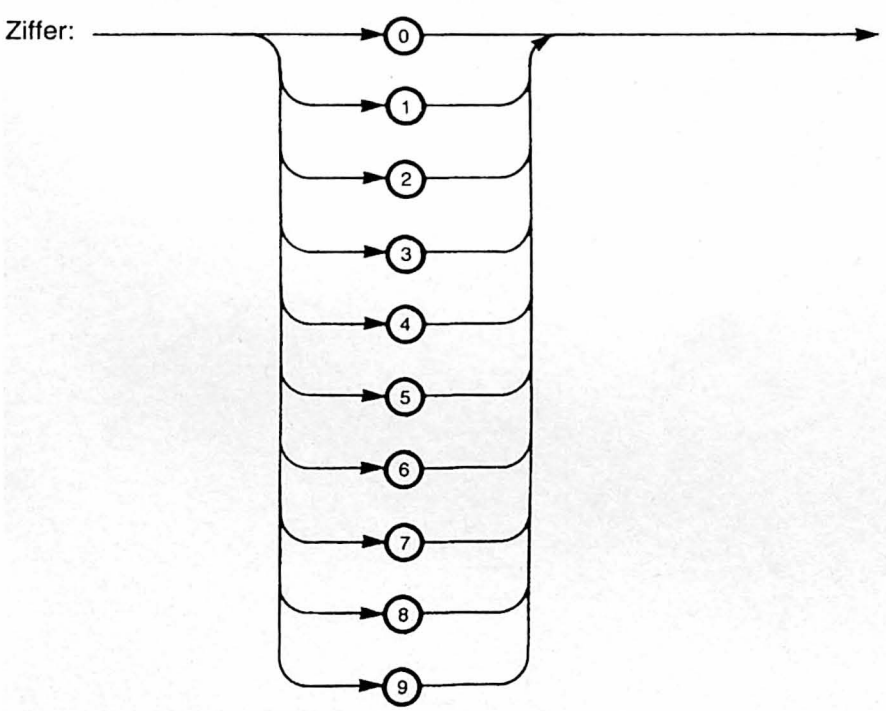
Ähnlich wie in DUCK ist auch in Pascal alles bestens geregelt.

Für alle möglichen Konstruktionen gibt es Syntaxdiagramme. ***Syntaxdiagramme***

Syntaxdiagramme werden gelesen, indem man dem Pfeil folgt. Oft sind mehrere Wege möglich, ein anderes Mal nur bestimmte. Achten Sie also auf den Pfeil. Ein Syntaxdiagramm sieht aus wie das Gleissystem einer Modelleisenbahn.

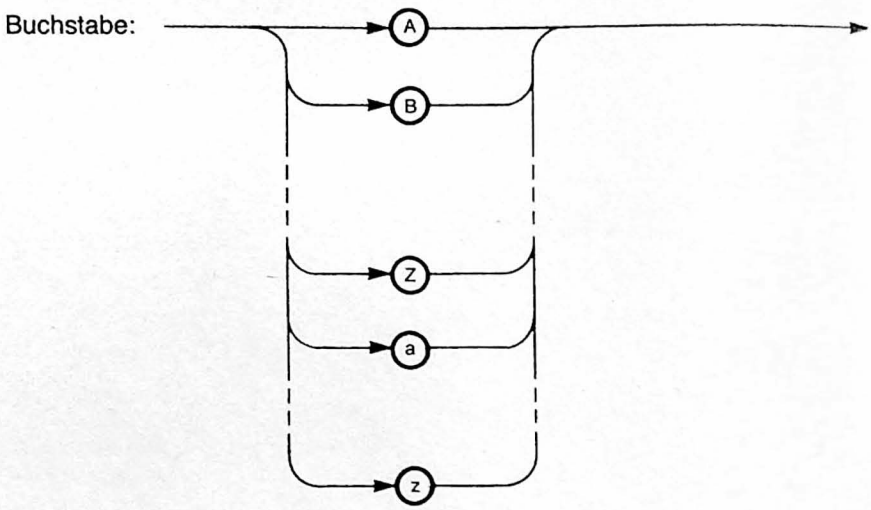
In den abgerundeten Kästen stehen fest definierte Symbole oder reservierte Wörter, in den eckigen Kästen Bezeichner, die durch weitere Syntaxdiagramme erklärt werden.

Einige Beispiele für grundlegende Pascal-Konstruktionen:



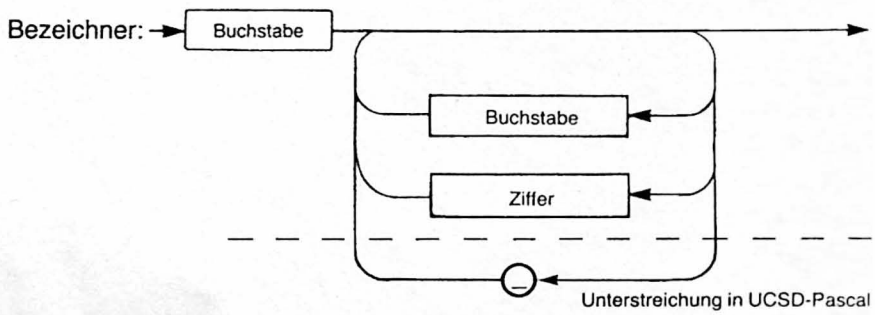
Ziffer

Buchstabe



Die Syntaxdiagramme für „Ziffer" und „Buchstabe" lassen nur jeweils einen Weg zu, d.h. eine Ziffer (ein Buchstabe) besteht aus genau einem der aufgeführten Symbole.

Bezeichner



Im Weg durch das Syntaxdiagramm „Bezeichner" müssen wir als erstes einen Buchstaben benutzen. Danach sind wir entweder fertig oder können noch eine(n) oder mehrere Ziffern (Buchstaben) oder Unterstreichungszeichen folgen lassen.

Beispiele Folgende Bezeichner sind also syntaktisch korrekt:

```
Bezeichner
BEGIN
ANFANG
x1
Drei45Sechs
Was_sonst       (in UCSD)
```

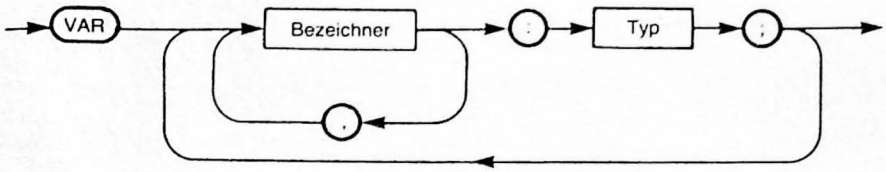
Falsch dagegen sind:

```
_4Teile
Was-sonst
_*Test*
```

Warum?

Variablen-deklaration Anweisung

Variablendeklaration:



Variablen-deklaration

Bemerkung: Typ muß natürlich als Syntaxdiagramm vorher erklärt sein (siehe Anhang).

So sind syntaktisch korrekte Variablendeklarationen: ***Beispiele***

```
VAR A, Test, Wert : INTEGER;
```

oder

```
VAR X : REAL;
    Y, Z : CHAR;
    Drucker : INTERACTIVE;
```

Falsch dagegen sind:

```
VARIABLES X, Y : STRING;
```

oder

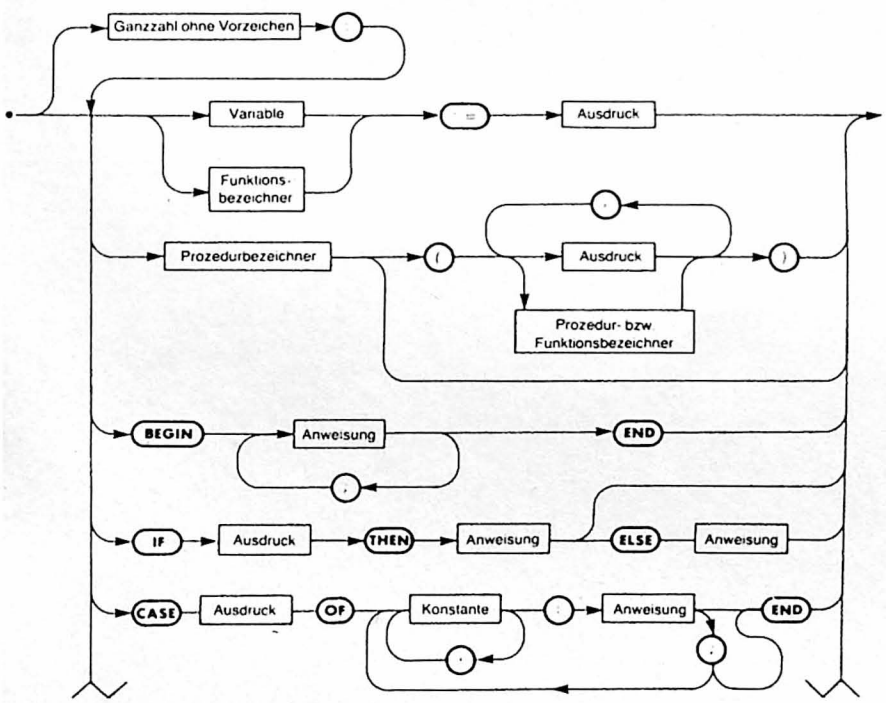
```
VAR: A, B, C : CHAR;
```

oder

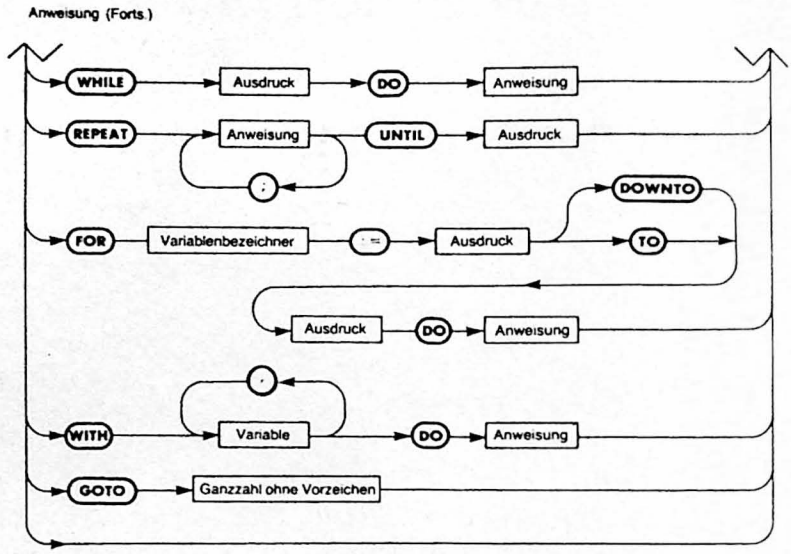
```
VAR X = REAL;
```

Warum?

Anweisung:



Anweisung



Beispiele Beispiele für korrekte Anweisungen:

```
A:=7;
```

oder

```
BEGIN
   I:=I+1;
   WRITELN(I)
END;
```

Wir werden in den folgenden Kapiteln noch einige Syntaxdiagramme kennenlernen. Insbesondere sei auf den Anhang verwiesen, dessen Übersicht über Syntaxdiagramme in Zweifelsfällen zu Rate gezogen werden kann.

Einfache Datentypen und deren Ein-/Ausgabeformate

Kapitel 5

Im Deklarationsteil eines Programms sind wir aufgefordert zu erklären, von welchem Datentyp die einzelnen Variablen sind. Dazu werden in diesem Kapitel die vier einfachen (skalaren) Datentypen INTEGER, REAL, CHAR und BOOLEAN dargestellt. Einfach heißt, daß diese Datentypen nicht aus anderen Typen zusammengesetzt sind.

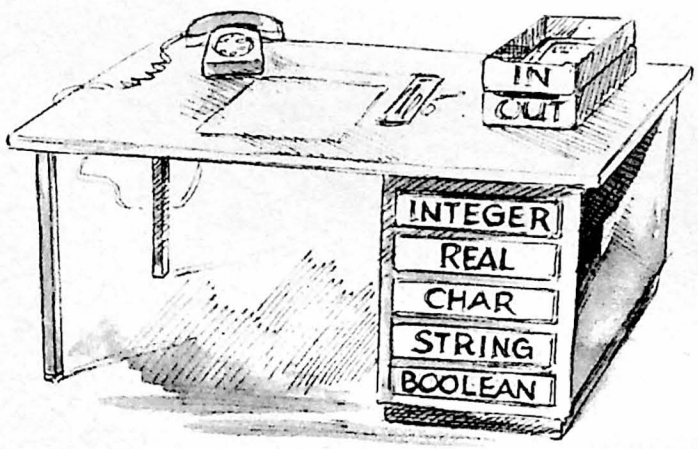
Außerdem wird der Datentyp STRING vorgestellt, der zwar nicht skalar ist, jedoch für unsere weiteren Programme von großem Nutzen ist.

Bei den hier beschriebenen Datentypen handelt es sich um eingebaute, sogenannte Standard-Datentypen. Weiterhin gibt es noch vom Benutzer selbst zu definierende Datentypen, die jedoch erst in Band 2 behandelt werden.

Zu den Datentypen werden die jeweiligen Ein- und Ausgabemöglichkeiten beschrieben.

Nach dem Lesen dieses Kapitels

- kennen wir die Syntax der Datentypen: ganze Zahl (INTEGER), Dezimalzahl (REAL), Zeichen (CHAR), Zeichenkette (STRING) und Wahrheitswert (BOOLEAN);
- können wir Variablen der dargestellten Datentypen im Zusammenhang mit den Standardprozeduren zur Ein- und Ausgabe richtig behandeln;
- kennen wir die formatierte Ausgabe;
- beherrschen wir arithmetische Operationen mit diesen Datentypen;
- können wir Ordnungsrelationen anwenden;
- können wir INTEGER und REAL sowie CHAR und STRING unterscheiden;
- erkennen und vermeiden wir Typkonflikte.



5.1 Ganze Zahlen – INTEGER

Zahlen

4711 5 22' 17" 14a
123456789101112 4 −3428
3.7 52,148 +21522

Das ist eine Reihe von Nummern.

Ganze Zahlen

Einige davon sind vom Datentyp „ganze Zahl", nämlich:

4711 , 4 , −3428 , +21522 .

Weiterhin ist 123456789101112 vom Datentyp „lange ganze Zahl".

Frage: Warum sind die anderen Nummern nicht ganzzahlig? – Was sind das für Objekte?

Aufgabe (Division mit Rest)

Wir kennen alle aus unserem Anfangs-Mathematikunterricht die Division ganzer Zahlen mit Rest. Dazu wollen wir ein kleines Programm schreiben: Eingegeben werden zwei ganze Zahlen, der Dividend und der Divisor. Ausgegeben wird das ganzzahlige Ergebnis und der Rest.

Blockdiagramm

Programm TEILE
Variablen Dividend,Divisor, Ergebnis, Rest : INTEGER
Anfang
Einlesen der Werte Dividend und Divisor
Berechne Ergebnis
Berechne Rest
Ausgabe Ergebnis und Rest
Ende.

INTEGER

Alle Variablen des Programms sollen GANZE ZAHLEN sein. Wir nennen diesen Datentyp in Pascal INTEGER.

Eine ganze Zahl zeichnet sich dadurch aus, daß sie einen Nachfolger und einen Vorgänger hat, der durch Addition (Subtraktion) mit 1 ermittelt wird.

Wertebereich MAXINT

Aufgrund der Darstellung im Rechner ist allerdings darauf hinzuweisen, daß es nur einen begrenzten Bereich von ganzen Zahlen gibt.

Die größte ganze Zahl ist durch die Konstante MAXINT und die kleinste durch −MAXINT-1 bestimmt. MAXINT ist eine vordefinierte Konstante, die rechnerabhängig ist. Im UCSD-Pascal ist MAXINT = $2^{15} - 1 = 32767$.

Ganze Zahlen

Folgende Darstellungen sind gültige INTEGER-Werte:

gültige INTEGER-Werte

```
12345
+24
−24
0
3
MAXINT
−MAXINT
```

Fehlerhafte Werte für INTEGER sind:

ungültige INTEGER-Werte

2.56	(kein Dezimalpunkt erlaubt)
3,456	(kein Komma erlaubt)
3E20	(keine Zehnerpotenz-Darstellung erlaubt)
120340	(größer als MAXINT)

Nachdem wir nun wissen, welche Form INTEGER-Variablen (und natürlich auch Konstanten, siehe Kap. 3.3) haben, brauchen wir noch Rechenoperationen:

Rechenoperationen

+	: Addition
−	: Subtraktion
*	: Multiplikation
DIV	: Division für ganze Zahlen mit ganzzahligem Ergebnis. Der mögliche Nachkommawert wird abgeschnitten (nicht gerundet!).
MOD	: Rest (modulo). Ergibt den Rest bei einer Division.

Beispiele:

Beispiele

```
3+5 ergibt 8
8−5 ergibt 3
12−20 ergibt −8
2*13 ergibt 26
7 DIV 2 ergibt 3
7 MOD 2 ergibt 1
13 MOD 5 ergibt 3
```

Wir können Zahlen miteinander vergleichen. Hierzu verwenden wir die folgenden Vergleichsoperatoren:

Vergleichsoperatoren

>	größer als
>=	größer oder gleich
<	kleiner als
<=	kleiner oder gleich
=	gleich
<>	ungleich

Nun läßt sich unser kleines Rechenprogramm verwirklichen. Dividend DIV Divisor ergibt das Ergebnis und Dividend MOD Divisor den Rest.

Programm

```
PROGRAM Teile;

  VAR Dividend, Divisor, Ergebnis, Rest : INTEGER;

BEGIN
  WRITE ('Eingabe des Dividenden: ');
  READLN (Dividend);
  WRITE ('Eingabe des Divisors  : ');
  READLN (Divisor);
  Ergebnis := Dividend DIV Divisor;
  Rest := Dividend MOD Divisor;
  WRITELN (Dividend:5,':',Divisor:5,'=',Ergebnis:5,' Rest',Rest:5)
END.
```

Bei einer Eingabe von 29 als Dividend und 3 als Divisor antwortet das Programm:

```
29:     3=     9 Rest     2
```

Ausgabeformate

Bei der Ausgabe der Werte fällt auf, daß eine formatierte Ausgabe benutzt wurde (siehe auch Kap. 3.2).

Die ganze Zahl hinter dem Doppelpunkt gibt an, mit wie vielen Stellen die Variable ausgedruckt werden soll.

Standardfunktionen

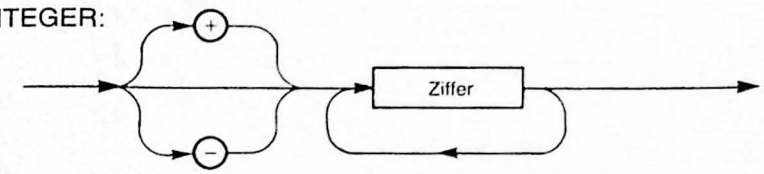
Im Zusammenhang mit dem Datentyp INTEGER sind noch einige Funktionen zu nennen, die verwendet werden können. Hinter einer Funktion steckt eine Reihe von Operationen, die mit dem Argument der Funktion (der Wert in der Klammer hinter dem Funktionsnamen) durchgeführt werden. Eine Funktion hat stets ein Ergebnis.

Im folgenden ist „i" eine INTEGER-Variable und „r" eine Variable vom Typ REAL (Dezimalzahl).

ABS(i)	Absolutwert einer Zahl i. Ergebnis vom Typ INTEGER. Beispiel: ABS(−10) ergibt 10 ABS(+10) ergibt 10
SQR(i)	Quadrat einer Zahl i. Ergebnis vom Typ INTEGER. Beispiel: SQR(3) ergibt 9
TRUNC(r)	Ganzzahliger Anteil einer Dezimalzahl (abgeschnitten, nie aufgerundet). Ergebnis vom Typ INTEGER. Beispiel: TRUNC(3.6) ergibt 3 TRUNC(−20.2) ergibt −20
ROUND(r)	Gerundeter ganzzahliger Teil einer Dezimalzahl. Ergebnis vom Typ INTEGER. Beispiel: ROUND(3.6) ergibt 4

INTEGER

Definition INTEGER



VAR i : INTEGER;

Variablen vom Datentyp INTEGER können die Werte −MAXINT-1,...,−3,−2,−1,0,1,2,3,...,MAXINT enthalten. Konstanten aus dieser Wertemenge sind ebenfalls vom Typ INTEGER.
MAXINT ist eine vordefinierte Konstante (in UCSD-Pascal: 32767),
Die Rechenoperationen sind: + (Addition), − (Subtraktion),
* (Multiplikation), DIV (Division), MOD (Rest).
Vergleichsoperatoren sind: < , <= , > , >= , = , <> .
Standardfunktionen mit einem Ergebnis vom Typ INTEGER sind:
ABS(i), SQR(i), TRUNC(r), ROUND(r).

Ausgabeformate: ***Ausgabeformate***

mit

```
VAR i,n : INTEGER;
```

ergibt:

```
WRITELN (i);    Ausgabe von i mit der tatsächlichen
                Stellenzahl von i
WRITELN (i:n);  Ausgabe von i mit n Stellen rechtsbündig
```

Besonderheit in UCSD-Pascal (Long INTEGER): ***Long INTEGER***

Schachbrett

Nach einer alten Legende soll ein brahmanischer Weiser für einen indischen König das Schachspiel erfunden haben. Der König soll dem Brahmanen dafür eine Belohnung angeboten haben, die der Weise selber wählen durfte. Dieser wünschte sich daraufhin: auf das erste Feld des Schachbretts ein Weizenkorn und auf die folgenden jeweils die doppelte Anzahl der Körner des vorhergehenden Feldes. Der König fand diesen Wunsch überaus bescheiden. Jedoch war er nach Berechnung der Anzahl aller Weizenkörner nicht in der Lage so viel Weizen aufzutreiben. Warum?

Wir wollen unseren Computer befragen. Dazu müßten wir ein Programm schreiben, das die Zahl $2^{64}-1$ berechnet und ausgibt. Dabei ergeben sich aber zwei kleine Probleme:

1. Die größte Zahl, die wir ausgeben können, ist 32767 (unsere Anzahl wird sicher größer werden!).
2. Pascal kennt keine Operation oder Funktion, die das Potenzieren übernimmt.

Abhilfe: In UCSD-Pascal gibt es den Datentyp Long INTEGER, der in der Variablendeklaration definiert wird, indem hinter dem Wörtchen INTEGER in eckigen Klammern die maximale Stellenzahl der Variablen (höchstens 36) angegeben wird.

Long INTEGER

Das Potenzieren realisieren wir durch eine Schleife (Definition siehe Kap. 6.1) von 2 bis 64.

Das Programm:

Programm

```
PROGRAM Schachkorn;
VAR Anz, Summe : INTEGER[30]; (* Anzahl der Koerner *)
    i : INTEGER;              (* Zaehlvariable *)
BEGIN
  Anz:=1;
  Summe:=1;
  WRITELN('Weizenkoerner auf Schachfeld');
  WRITELN;
  WRITELN('auf Feld Nr. 1 ist :',Anz:30,' Korn.');
  FOR i:=2 TO 64 DO
    BEGIN
      Anz:=Anz*2;
      WRITELN('auf Feld Nr.',i:2,' sind:',Anz:30,' Koerner.');
      Summe:=Summe+Anz
    END;
  WRITELN;
  WRITELN('auf allen Feldern sind:',Summe:27,' Koerner')
END.
```

Programmfunktion

Funktion des Programms: Die Anzahl für das erste Feld wird mit 1 festgelegt. Die Schleife (FOR i:=2 TO 64 DO...) berechnet die Anzahl der Körner auf den folgenden Feldern. Mit der Variablen Summe wird die Anzahl der Körner auf den Feldern aufsummiert.

Die Ausgabe des Programms:

Ausgabe

```
Weizenkoerner auf Schachfeld

auf Feld Nr. 1 ist :                          1 Korn.
auf Feld Nr. 2 sind:                          2 Koerner.
auf Feld Nr. 3 sind:                          4 Koerner.
auf Feld Nr. 4 sind:                          8 Koerner.
auf Feld Nr. 5 sind:                         16 Koerner.
```

```
auf Feld Nr.57 sind:            72057594037927936 Koerner.
auf Feld Nr.58 sind:           144115188075855872 Koerner.
auf Feld Nr.59 sind:           288230376151711744 Koerner.
auf Feld Nr.60 sind:           576460752303423488 Koerner.
auf Feld Nr.61 sind:          1152921504606846976 Koerner.
auf Feld Nr.62 sind:          2305843009213693952 Koerner.
auf Feld Nr.63 sind:          4611686018427387904 Koerner.
auf Feld Nr.64 sind:          9223372036854775808 Koerner.

auf allen Feldern sind:      18446744073709551615 Koerner
```

Einschränkungen

Achtung: Bei dem Datentyp Long INTEGER sind einige Besonderheiten zu beachten.

Die Operationen + , − , * , DIV gelten auch für Long INTEGER. Das Ergebnis ist allerdings dann auch Long INTEGER!

Normale und lange INTEGER-Zahlen können zwar gemischt werden, jedoch kann niemals einer INTEGER-Variablen ein Wert vom Typ Long INTEGER zugewiesen werden.

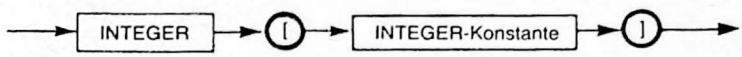
Frage: Warum?

INTEGER ↔ Long INTEGER

Soll eine Long INTEGER-Zahl i in INTEGER umgewandelt werden, so verwenden wir TRUNC(i). Voraussetzung ist aber, daß die Zahl i kleiner als MAXINT und größer als −MAXINT ist!

Definition Long INTEGER

Long INTEGER:



VAR I : INTEGER[n]; mit 0<n<37

INTEGER-Variable mit n Stellen. Alle INTEGER-Operationen außer MOD sind erlaubt. Vorsicht bei Wertzuweisungen! Umwandlung von Long INTEGER in INTEGER mit TRUNC.

Übungen:

5.1.1 Berechnen Sie selbst:

```
14 MOD 3
 4 DIV 2
 4 / 2
28 DIV 5
28 / 5
TRUNC (13/5)
ROUND (13/5)
```

5.1.2 Schreiben Sie ein Programm, das Zahlen (kleiner als MAXINT) in Zehntausender, Tausender, Hunderter, Zehner und Einer zerlegt.

5.1.3 Schreiben Sie ein Programm, das eine maximal 15stellige Binärzahl (Zahl im Zweiersystem) in eine ganze Zahl umwandelt.

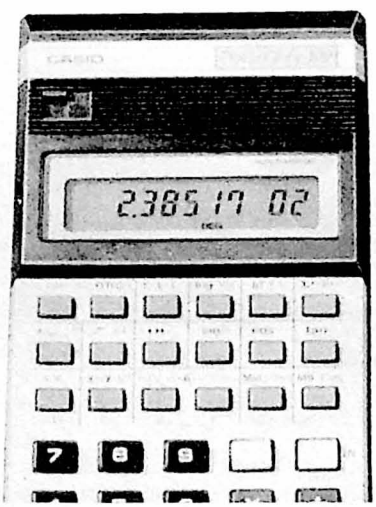
Beispiel: 1101 (Bin) = 13 (Dez).

Berechnen Sie auch die Binärzahl, die aus 15 (8) Einsen besteht.

Was fällt Ihnen auf?

Tip: Mit einer FOR-Schleife alle 15 Stellen auf Eins oder Null untersuchen!

5.2 Dezimalzahlen – REAL



Aufgabe (Kugel)

Im folgenden Programm wollen wir Oberfläche und Volumen einer Kugel berechnen. Dazu kennen wir aus der Geometrie die Formeln:

$$O = 4*\pi*r^2 \quad \text{und} \quad V = 4/3*\pi*r^3$$

Die Programmiersprache Pascal sieht keine vordefinierte Konstante für π vor. Daher müssen wir sie selbst definieren: pi=3.1416.

Blockdiagramm

Programm KUGEL
Konstante pi=3.1416
Variablen Radius, Oberflaeche, Volumen : REAL
Anfang
Eingabe des Wertes für Radius
Oberflaeche:=4*pi*Radius^2
Volumen:=4/3*pi*Radius^3
Ausgabe der Ergebnisse mit 2 Stellen hinter dem Komma
Ende.

REAL

Die Konstante ist durch ihre Form eindeutig vom Typ Dezimalzahl. In Pascal wird dieser Datentyp REAL genannt. Die Variablen sollen auch alle von diesem Typ sein.

Wertebereich

Auch dieser Datentyp hat einen beschränkten Wertebereich. Die Ober- und Untergrenze dieses Wertebereichs ist allerdings sehr vom verwendeten Rechner abhängig. Daher siehe Handbuch!

Genauigkeit

Außerdem hapert es etwas mit der Genauigkeit. So läßt sich natürlich 1/3 nicht durch 0.33333333333333... darstellen, da die Ziffer 3 noch unendlich oft auftreten würde. So muß also ab einer bestimmten Stelle gerundet werden. Dies führt bei vielen Rechenschritten möglicherweise zu erheblichen Rundungsfehlern. Die Rechengenauigkeit an sich (d.h. die Anzahl der Stellen hinter dem Dezimalpunkt) hängt ebenfalls von der benutzten Rechenanlage ab.

Gleichheit?

Achtung: Um unnötigen Ärger zu vermeiden, sollten wir folgende Hinweise beim Arbeiten mit REAL-Variablen beachten:

- Niemals die Gleichheit von Rechenergebnissen abfragen, sondern überprüfen, ob der Absolutwert der Differenz der Ergebnisse einen bestimmten Wert unterschreitet.
- Die Subtraktion fast gleich großer REAL-Zahlen vermeiden, wo es möglich ist.

Rundungsfehler

- Möglichst wenig Rechenschritte, um den Rundungsfehler klein zu halten.

Wie sehen nun Zahlen des Typs REAL aus?

Mit einem möglichen Vorzeichen schreiben wir Ziffern vor und hinter dem Dezimalpunkt wie bei Taschenrechnern.

Rechenoperationen

Außerdem ist die Darstellung mit Zehnerpotenzen erlaubt. So kann man 3400 darstellen als $3.4 * 10^3$; also schreiben als 3.4E3 (ebenfalls wie bei Taschenrechnern).

Gültige REAL-Zahlen sind z. B.: ***gültige REAL-Zahlen***

12.
12.0
+3.67 oder 3.67
−9.03
0.45
4.67E+3 oder 4.67E3
1.4E−4

Bei vielen Rechnern ebenfalls erlaubt:

.5 statt 0.5
15 statt 15. oder 15.0 (hier wandelt der Rechner eine INTEGER-Zahl selbst in REAL um)

Die Rechenoperationen für REAL sind: ***Rechenoperationen***

\+ : Addition

− : Subtraktion

* : Multiplikation

/ : Division mit Ergebnis vom Typ REAL

Außerdem sind die schon bekannten Vergleichsoperatoren

> , >= , < , <= , = , <> (siehe INTEGER, Kap. 5.1)

zu verwenden.

Nun unser kleines Programm:

Programm

```
PROGRAM Kugel;

CONST pi=3.1416;

VAR Radius, Oberflaeche, Volumen : REAL;

BEGIN
  WRITE ('Eingabe des Radius: ');
  READLN (Radius);
  Oberflaeche := 4 * pi * Radius * Radius;
  Volumen := 4/3 * pi * Radius * Radius * Radius;
  WRITELN ('Die Kugel mit dem Radius ',Radius:6:2,' hat ');
  WRITELN ('ein Volumen von ',Volumen:9:2);
  WRITELN ('und eine Oberflaeche von ',Oberflaeche:7:2)
END.
```

Auch in diesem Programm wurde die formatierte Ausgabe gewählt (siehe auch Kap. 3.2). ***Ausgabeformate***

Die ganze Zahl hinter dem ersten Doppelpunkt gibt die gesamte Anzahl der Stellen an (einschließlich Dezimalpunkt!).

Die ganze Zahl hinter dem zweiten Doppelpunkt gibt die Zahl der Stellen hinter dem Dezimalpunkt an.

Standard Funktionen

Standard-funktionen

Hier noch die Standardfunktionen für den Datentyp REAL:
(r steht für eine Variable vom Typ REAL, i für INTEGER und x für INTEGER oder REAL)

ABS(r)	Absolutwert einer Zahl r. Ergebnis REAL.
SQR(r)	Quadrat von r. Ergebnis REAL.
SIN(x)	Sinus von x (x im Bogenmaß). Ergebnis REAL.
COS(x)	Kosinus von x (x im Bogenmaß). Ergebnis REAL.
ATAN(x)	Arcustangens von x (im Bogenmaß). Ergebnis REAL. In Standard-Pascal ARCTAN(x).
LN(x)	Natürlicher Logarithmus von x. Ergebnis REAL.
EXP(x)	Exponentialfunktion e^x. Ergebnis REAL.
SQRT(x)	Quadratwurzel von x. Ergebnis REAL.

In UCSD-Pascal zusätzlich:

PWROFTEN(i)	Zehnerpotenz 10^i mit 0<=i<=37. Ergebnis REAL.
LOG(x)	Dekadischer Logarithmus von x. Ergebnis REAL.

INTEGER ↔ REAL

Unterschied INTEGER ↔ REAL:

Bei folgender Variablendeklaration:

```
VAR x,y : REAL;
      i : INTEGER;
```

ergeben folgende Programmzeilen einen Fehler:

```
x:=1.5;
y:=2;
i:=x*y; (* hier fehlerhafte Zuweisung *)
```

Typkonflikt

Hier liegt ein sogenannter Typkonflikt vor. Obwohl in diesem Fall das Ergebnis der Multiplikation als ganze Zahl dargestellt werden kann, ist es doch von der Typdeklaration her REAL. x*y ergibt nämlich nicht 3, sondern 3.0.

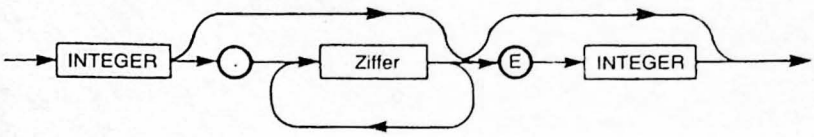
Einen Ausweg aus diesem Dilemma bieten die Übergangsoperationen von REAL nach INTEGER:

TRUNC(r) und ROUND (r) (siehe Kap. 5.1).

REAL

Definition REAL

REAL:



VAR r : REAL;

Variablen vom Datentyp REAL sind Dezimalzahlen der Form:
3.6 oder +3.6
0.7
−4.5E6 oder −4.5E+6

Der Wertebereich und die Rechengenauigkeit sind von der benutzten Rechenanlage abhängig. Rundungsfehler treten auf!

Die Rechenoperationen sind: + (Addition), − (Subtraktion), * (Multiplikation), / (Division).
Vergleichsoperatoren sind: < , <= , > , >= , = , <> .

Standardfunktionen mit einem Ergebnis vom Typ REAL sind: ABS(r), SQR(r), SIN(x), COS(x), ARCTAN(x), LN(x), EXP(x), SQRT(x).
In UCSD-Pascal: ATAN(x) statt ARCTAN(x) und zusätzlich PWROFTEN(i), LOG(x).

Ausgabeformate:

Ausgabeformate

mit:

```
VAR r : REAL;
    n,m : INTEGER;
```

ergibt:

WRITELN (r);	Ausgabe von r mit größter Genauigkeit (in der Regel mit der Zehnerpotenzschreibweise).
WRITELN (r:n:m);	Ausgabe von r mit insgesamt n Stellen (einschließlich Dezimalpunkt!), davon m Stellen hinter dem Dezimalpunkt.

Übungen:

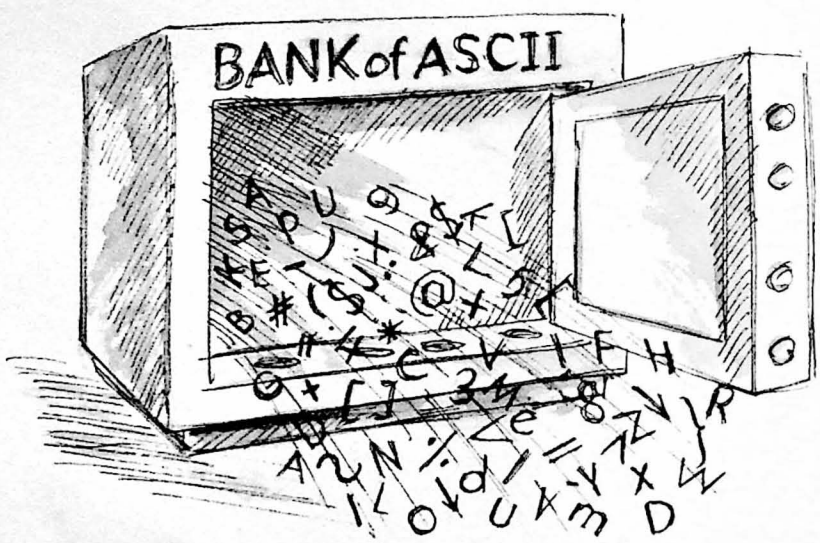
5.2.1 Schreiben Sie ein Programm, das nach Eingabe der Grundseite und der Höhe den Flächeninhalt eines Dreiecks berechnet.

5.2.2 Zeichnen Sie (mit Hilfe der Grafik) einen Kreis auf dem Bildschirm, unter Verwendung der Kreisgleichung $x^2 + y^2 = r^2$ (mit r:Radius).

5.2.3 Für einen Quader werden die Daten: Breite, Höhe und Länge angegeben. Ein Programm soll Oberfläche und Volumen sowie die Längen der Flächendiagonalen und die Länge der Raumdiagonale berechnen.

5.2.4 Zeigen Sie, daß sich die Funktion ROUND(x) durch die TRUNC-Funktion ersetzen läßt.

5.3 Zeichen – CHAR



Zeichensatz

Unser Computer kann nicht nur Zahlen (numerische Ausdrücke) verarbeiten, sondern auch Zeichen. Unter Zeichen verstehen wir Buchstaben, Ziffern, Satzzeichen, Zeichen für Rechenoperationen und sogenannte Steuerzeichen (z. B. für den Drucker).

ASCII-Tabelle (Anhang)

Der Datentyp, der für diese alphanumerischen Zeichen verwendet wird, heißt CHAR. Welche Zeichen zum Zeichensatz gehören, hängt vom Rechnertyp ab. Sehr viele Rechner verwenden Zeichen nach dem sogenannten ASCII-Zeichensatz (American Standard Code for Information Interchange). Dieser Zeichensatz umfaßt in der Regel 128 Zeichen (manche Rechner haben einen erweiterten Zeichensatz von 256 Zeichen).

Aufgabe: Sehen sie sich den ASCII-Zeichensatz im Anhang an!

Nummern

Die Zeichen sind durchnumeriert von 0 bis 127. Wir werden gleich auf die Bedeutung der Nummern zu sprechen kommen. Zu beachten ist, daß die Zeichen von Nummer 0 bis 31 nur sogenannte Steuerzeichen sind, also Zeichen, die eine bestimmte Funktion ausüben. So ist z. B. Nummer 12 das Zeichen FF (Form-Feed) – das ist ein Seitenvorschub (auf dem Bildschirm oder dem Drucker). Die restlichen

Zeichen mit den Nummern 32 bis 126 sind sichtbare Zeichen. Nur Nummer 127 hat noch eine Steuerfunktion (Löschzeichen).

Die Nummern der Zeichen haben eine große Bedeutung. Durch sie ist der Zeichensatz geordnet, das Zeichen A ist z. B. kleiner als das Zeichen B.

Ordnung

In Pascal werden Zeichen immer zwischen einfache Anführungsstriche (Hochkommata) gesetzt. Will man einen Anführungsstrich als Zeichen verwenden, so wird er doppelt zwischen zwei Hochkommata geschrieben, d. h. ''''.

Einige Beispiele:

Beispielprogramm

```
PROGRAM Testzeichen;

  CONST a='A';
        b='B';

  VAR ch:CHAR;

  BEGIN
    READ (ch);
    WRITELN (ch:4);
    READLN (ch);
    WRITELN (ch);
    WRITELN (a,b);
    ch := 'C';
    WRITELN (ch);
    ch := a;
    WRITELN (ch)
  END.
```

Bei der ersten Eingabe (mit READ) reicht es, wenn der Benutzer eine Taste drückt (ohne Return-Taste). Die zweite Eingabe (mit READLN) muß mit der Return-Taste abgeschlossen werden.

In der formatierten Ausgabe gibt die ganze Zahl hinter dem Doppelpunkt die Anzahl der Stellen an, die das Zeichen beansprucht. Ist die Stellenzahl größer als 1, so werden Leerzeichen vorangestellt.

STRING ↔ CHAR

Achtung: Die Zuweisung ch:= 'AB' wäre falsch, da 'AB' aus mehr als einem Zeichen besteht (es ist vom Typ STRING, siehe Kap. 5.4).

Standardfunktionen

Mit dem Typ CHAR lassen sich natürlich keine arithmetischen Operationen ausführen. Trotzdem gibt es einige Standardfunktionen:

(c ist vom Typ CHAR und i vom Typ INTEGER)

ORD(c)	Ordnungsnummer von c, d. h. die Nummer des Zeichens in der Codetabelle (z. B. ASCII). Ergebnis vom Typ INTEGER. Beispiel: ORD('A') ergibt 65
CHR(i)	Zeichenfunktion. Liefert das Zeichen mit der Nummer i aus der Codetabelle (z. B. ASCII). Ergebnis vom Typ CHAR. Beispiel: CHR(65) ergibt 'A'

PRED(c) Vorgängerfunktion. Liefert das dem Zeichen c in der Codetabelle vorangehende Zeichen. Ergebnis vom Typ CHAR.
Beispiel: PRED('E') ergibt 'D'

SUCC(c) Nachfolgerfunktion. Liefert das dem Zeichen c in der Codetabelle nachfolgende Zeichen. Ergebnis vom Typ CHAR.
Beispiel: SUCC('E') ergibt 'F'

Außerdem gelten die schon bekannten Vergleichsoperatoren:

< , <= , > , >= , = , <> .

Ordnung

Dabei ist zu beachten, daß die Elemente des Zeichensatzes in der Reihenfolge der Codetabelle angeordnet sind.

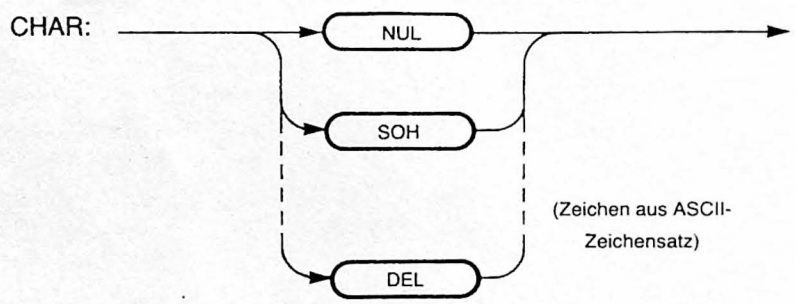
Im ASCII-Zeichensatz gilt z. B.:

...' ' < '!' < '"' < '#' <...< '/' < '1' < '2' <...< 'A' <...< 'Z' <...

Achtung bei Sortierproblemen:

Einige Sonderzeichen kommen in der Reihenfolge vor den Buchstaben. Insbesondere das Leerzeichen hat mit 32 die kleinste Ordnungsnummer der sichtbaren Zeichen.

Definition CHAR



VAR c : CHAR;

Variablen vom Datentyp CHAR sind alphanumerische Zeichen. Die Zeichen müssen in einfachen Anführungsstrichen stehen. Der Zeichensatz ist von der benutzten Rechenanlage abhängig. Es besteht eine Ordnung innerhalb des Zeichensatzes, so daß die Vergleichsoperatoren < , <= , > , >= , = , <> verwendet werden können.
Standardfunktionen mit einem Ergebnis vom Typ CHAR sind:
CHR(i), PRED(c), SUCC(c).
Standardfunktion mit einem Ergebnis vom Typ INTEGER ist:
ORD(c).

Eingabe- und Ausgabeformate:
mit

Formate

```
VAR c : CHAR;
    n : INTEGER;
```

ergibt:

READ (c);	Eingabe eines Zeichens ohne Return-Taste.
READLN (c);	Eingabe eines Zeichens mit Return-Taste.
WRITELN (c);	Ausgabe mit einer Stelle.
WRITELN (c:n);	Ausgabe mit n Stellen. Die führenden Stellen werden mit Leerzeichen aufgefüllt.

Übungen:

5.3.1 Schreiben Sie ein Programm, das nach Eingabe einer Nummer das dazugehörige ASCII-Zeichen ausgibt.

5.3.2 Umgekehrt! Schreiben Sie ein Programm, das nach Eingabe eines ASCII-Zeichens die dazugehörige Nummer ausgibt.

5.3.3 Erweitern Sie das Programm aus 5.3.2 dahingehend, daß zusätzlich der Vorgänger und der Nachfolger des eingegebenen Zeichens ausgegeben wird.

5.3.4 Geheimschrift: Schreiben Sie ein Programm, das einen eingegebenen Text zeichenweise nach einem von Ihnen gewählten Muster verschlüsselt und ausgibt.

5.3.5 Ein zweites Programm (zu 5.3.4) entschlüsselt einen eingegebenen Text nach dem gleichen Muster.

Tip: Mit einer REPEAT...UNTIL...-Schleife (Beispiel siehe Kap. 3.5) Zeichen eingeben, bis ein vereinbartes Ende-Zeichen, z. B. '/', eingegeben wird.

5.4 Zeichenketten – STRING in UCSD-Pascal

Standard-Pascal sieht den Datentyp STRING an sich nicht vor. Viele Pascal-Versionen besitzen ihn trotzdem. Hier werden die Standardfunktionen im UCSD-Pascal vorgestellt. Geringfügige Abweichungen in anderen Pascal-Versionen lassen sich aus dem Handbuch des Rechners ermitteln.

STRING in UCSD-Pascal

Der Datentyp STRING wird für Zeichenketten verwendet. Eine Zeichenkette ist eine Aneinanderreihung von Elementen des Typs CHAR.

Standard-Pascal

Daher kann man sich den Datentyp STRING selbst definieren, wenn er in der benutzten Pascal-Version nicht vorgesehen ist. Die Form ist dann:

```
TYPE STRING = PACKED ARRAY [1...n] OF CHAR;
```

wobei n die maximale Anzahl der Zeichen einer Zeichenkette ist. Die TYPE-Vereinbarung kommt vor der Variablendeklaration (siehe auch Band 2).

Der UCSD-Pascal-Benutzer braucht STRING nicht zu definieren, es ist schon vordefiniert. Die maximale Anzahl der Zeichen einer STRING-Variablen ist 80 Zeichen. Soll eine andere Länge vereinbart werden, so wird in der Variablendeklaration hinter das Wort STRING eine ganze Zahl zwischen 1 und 255 in eckigen Klammern geschrieben. Beispiel:

```
VAR s : STRING [15];
```

maximale Länge

Die maximale Länge der Zeichenkette s ist 15 Zeichen.

Als Beispiel zum Datentyp STRING wollen wir ein kleines Programm schreiben, das ein eingegebenes Wort untereinander schreibt. Dazu brauchen wir eine sogenannte Schleife (siehe Kap. 6.1), die von 1 bis zur Anzahl der Zeichen des Wortes zählt:

```
FOR i := 1 TO LENGTH(Wort);
```

Programm

Hier das Programm:

```
PROGRAM Worttest;

  VAR Wort : STRING;
      i : INTEGER;  (* Laufvariable *)

  BEGIN
    WRITELN ('Geben Sie ein Wort ein: ');
    READLN (Wort);
    FOR i := 1 TO LENGTH(Wort) DO
      WRITELN (Wort[i])
  END.
```

Zeichen aus STRING

In diesem Programm sehen wir, daß es möglich ist, ein Zeichen aus einer Zeichenkette herauszunehmen, indem wir die Nummer des Zeichens in eine eckige Klammer hinter den Namen der Zeichenkette schreiben.

Beispiel:

```
Wort := 'Computer';
Wort [3] ergibt 'm'
```

STRING ↔ CHAR

Eine Zeichenkette, die nur ein Zeichen enthält, ist damit noch lange nicht vom Typ CHAR. Eine Zuweisung oder ein Vergleich der Datentypen STRING und CHAR führt unweigerlich zu einem Typkonflikt.

Standard Funktionen

Die einzige Möglichkeit ist:

```
VAR c : CHAR;
    s : STRING;
    n : INTEGER;
```

und die Zuweisung:

```
c := s [n];
```

wobei n die Nummer des Zeichens im STRING s ist.

Standardfunktionen

Als Standardfunktion mit Zeichenketten haben wir im Programm LENGTH(s) benutzt. Es gibt noch weitere Funktionen (s, s1, s2 ... sind im folgenden vom Typ INTEGER):

LENGTH(s)	Länge der Zeichenkette s, d.h. Anzahl der Buchstaben. Ergebnis vom Typ INTEGER. Beispiel: LENGTH('Wort') ergibt 4
POS(s1,s)	Position des erstmaligen Auftretens der Zeichenkette s1 in der Zeichenkette. Ergebnis vom Typ INTEGER. Beispiel: POS('buch','Handbuch') ergibt 5 POS('ball','Handbuch') ergibt 0 POS('ei','Weinstein') ergibt 2
CONCAT (s1,s2,s3,...,sn)	Verkettung mehrerer Zeichenketten. Ergebnis vom Typ STRING. Beispiel: CONCAT('Hand','buch') ergibt 'Handbuch'
COPY(s,n,m)	Herausnehmen eines Teils aus der Zeichenkette s ab der Stelle n mit der Länge m. Ergebnis vom Typ STRING. Beispiel: COPY('Computer',4,3) ergibt 'put'

Standardprozeduren

Neben den Standardfunktionen gibt es noch die Standardprozeduren mit Zeichenketten. Im Gegensatz zu den Funktionen, die immer ein Ergebnis haben, das einer Variablen zugewiesen werden muß, werden Prozeduren nur aufgerufen und können dann Variablen ändern, die ihnen beim Prozeduraufruf mitgegeben werden (siehe auch Kap. 9.2).

INSERT(s1,s,n)	Einfügen einer Zeichenkette s1 in die Zeichenkette s an der Stelle n. Beispiel: Sei s:='Comer'; INSERT('put',s,4) ergibt 'Computer' für s.
DELETE(s,n,m)	Löschen von m Zeichen ab der Stelle n in der Zeichenkette s. Beispiel: Sei s:='Buchstaben'; DELETE(s,3,5) ergibt 'Buben' für s.
STR(l,s)	Wandelt eine Zahl l vom Typ Long INTEGER in einen String s um.

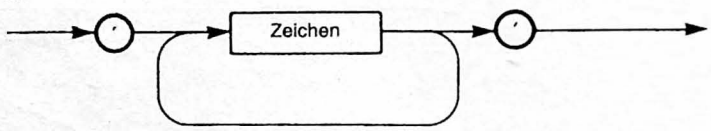
Da an dieser Stelle die Anwendung von Funktionen und Prozeduren noch nicht bekannt ist, folgt ein kleines Programm, das die Benutzung der o.g. Funktionen und Prozeduren verdeutlichen soll:

Programm

```
PROGRAM Stringdemo;

  VAR Wort, Teil : STRING;
      Stelle, Laenge : INTEGER;

  BEGIN
    Wort := 'Donauschiff';
    Laenge := LENGTH(Wort);
    WRITELN (Laenge);          (* Ausgabe: 11 *)

    Teil := 'aus';
    Stelle := POS(Teil,Wort);
    WRITELN (Stelle);          (* Ausgabe: 4 *)

    Wort := CONCAT(Wort,'skapitaen');
    WRITELN (Wort);            (* Ausgabe: Donauschiffskapitaen *)

    Teil := COPY(Wort,6,6);
    WRITELN (Teil);            (* Ausgabe: schiff *)

    INSERT('dampf',Wort,6);
    WRITELN (Wort);
        (* Ausgabe: Donaudampfschiffskapitaen *)

    Stelle := 1;
    Laenge := 17;
    DELETE(Wort,Stelle,Laenge);
    WRITELN (Wort)             (* Ausgabe: kapitaen *)

  END.
```

Definition STRING

STRING:



VAR s : STRING;
t : STRING [n]; mit 0<n<256

Variablen vom Datentyp STRING sind Zeichenketten. Die maximale Anzahl der Zeichen wird durch n angegeben. Wird n nicht angegeben, so ist n=80.
Da STRING als ARRAY [1..n] OF CHAR vordefiniert ist, kann über s[i] auf das i-te Zeichen von s zugegriffen werden (mit i vom Typ INTEGER). s[i] ist vom Typ CHAR.
Über den Datentyp CHAR ist der Typ STRING ebenfalls geordnet, so daß die Vergleichsoperatoren < , <= , > , >= , = , <> verwendet werden können.
Standardfunktionen mit einem Ergebnis vom Typ INTEGER sind: LENGTH(s) und POS(s1,s).
Standardfunktionen mit einem Ergebnis vom Typ STRING sind: CONCAT(s1,s2,...,sn) und COPY(s,n,m).
Standardprozeduren sind:
INSERT(s1,s,n), DELETE(s,n,m) und STR(l,s).

Ausgabeformate:

Ausgabeformate

mit

```
VAR s : STRING;
    n : INTEGER;
```

ergibt:

WRITELN(s);	Ausgabe von s mit LENGTH(s) Stellen.
WRITELN(s:n);	Ausgabe von s mit n Stellen. Wenn n größer als LENGTH(s) ist, so werden die führenden Stellen mit Leerzeichen aufgefüllt.

Übungen:

5.4.1 Worin liegt der Unterschied zwischen STRING und CHAR?

5.4.2 Schreiben Sie ein Programm, das ein beliebiges eingegebenes Wort rückwärts, untereinander und um jeweils ein Zeichen nach rechts versetzt schreibt.

5.4.3 Ändern Sie die Aufgaben 5.3.4 und 5.3.5 so, daß ein eingegebener Satz (vom Typ STRING) in Geheimschrift codiert (decodiert) wird.

5.5 Wahrheitswerte – BOOLEAN

DER UNTERE SATZ IST WAHR
DER OBERE SATZ IST GELOGEN

Was ist denn nun wahr???

Aus der Aussagelogik kennen wir sogenannte logische Aussagen. Sie können die Werte wahr oder falsch annehmen.

Aussagelogik

Beispiel:

„Paris ist die Hauptstadt von England" ist falsch.
„Paris ist die Hauptstadt von Frankreich" ist wahr.
„Dieses Buch ist leicht verständlich" ist objektiv nicht entscheidbar.

Alle drei Aussagen sind allerdings für unsere Arbeit mit dem Computer unbrauchbar. Insbesondere darf es niemals eine unentscheidbare Situation geben.

Vielmehr haben wir es mit Aussagen folgenden Typs zu tun:

Aussagen in Pascal

5=4 ist falsch.
5>4 ist wahr.

In Pascal haben die Wahrheitswerte folgende Namen:

TRUE – wahr
FALSE – falsch.

Außerdem kann man einen Wahrheitswert einer Variablen zuordnen, die dann vom Typ BOOLEAN ist.

BOOLEAN

Zuweisungen

Wenn die Variable b vom Datentyp BOOLEAN ist, dann ist die Zuweisung:

b := TRUE;	richtig
b := FALSE;	richtig
b := b = FALSE;	richtig
b := 17 < 3;	richtig
b := i <> j;	richtig (mit i,j vom Typ INTEGER)
b := (12>2) AND (7<2);	richtig
b := 12 + 3;	falsch.

Aufgabe (Wahrheitstabelle)

Wir wollen ein Programm schreiben, das eine Wahrheitstabelle für die Verknüpfung und (AND) angibt.

Blockdiagramm

Programm WAHR
Konstanten w='wahr', F='falsch'
Variablen a,b,c, : BOOLEAN
Anfang
Schreibe Tabellenkopf (a,b,a und b)
a:=wahr , b:=wahr , c:=a und b
Schreibe (W,W)
Wenn c wahr ist dann / andernfalls
schreibe (W) | schreibe (F)
a:=wahr , b:=falsch , c:=a und b
Schreibe (W,F)
Wenn c wahr ist dann / andernfalls
schreibe (W) | schreibe (F)
a:=falsch , b:=wahr , c:=a und b
Schreibe (F,W)
Wenn c wahr ist dann / andernfalls
schreibe (W) | schreibe (F)
a:= falsch , b:=falsch , c:=a und b
Schreibe (F,F)
Wenn c wahr ist dann / andernfalls
schreibe (W) | schreibe (F)
Ende.

Verknüpfungen

Es ergibt sich in UCSD-Pascal die Schwierigkeit, daß Daten vom Typ BOOLEAN weder direkt ein- noch ausgegeben werden können. Daher die Hilfskonstruktion:

Ausgabe in UCSD-Pascal

Wenn c wahr ist, dann schreibe wahr, andernfalls schreibe falsch. Dazu verwenden wir den Befehl IF...THEN...ELSE (siehe Kap. 7.2).

```
IF c=TRUE THEN WRITELN('wahr') ELSE WRITELN
('falsch');
```

Das Programm:

Programm

```
PROGRAM Wahr;

  CONST W=' wahr ';
        F='falsch';

VAR a,b,c : BOOLEAN;

BEGIN
  WRITELN ('    A          B         A und B');
  WRITELN ('-------------------------------');
  a := TRUE;   b := TRUE;
  c := a AND b;
  WRITE (W:9,W:9);
  IF c=TRUE THEN WRITELN (W:9) ELSE WRITELN (F:9);
  a := TRUE;   b := FALSE;
  c := a AND b;
  WRITE (W:9,F:9);
  IF c=TRUE THEN WRITELN (W:9) ELSE WRITELN (F:9);
  a := FALSE;  b := TRUE;
  c := a AND b;
  WRITE (F:9,W:9);
  IF c=TRUE THEN WRITELN (W:9) ELSE WRITELN (F:9);
  a := FALSE;  b := FALSE;
  c := a AND b;
  WRITE (F:9,F:9);
  IF c=TRUE THEN WRITELN (W:9) ELSE WRITELN (F:9)
END.
```

Wenn wir später (Kap. 9) Prozeduren kennen, läßt sich das Programm noch um einiges vereinfachen.

Werden Daten durch die bekannten Operatoren < , <= , > , >= , = , <> verglichen, so ist das Ergebnis vom Typ BOOLEAN (und könnte einer entsprechenden Variablen zugewiesen werden).

Vergleichsoperatoren

Folgende Operatoren für Ausdrücke vom Typ BOOLEAN können benutzt werden (mit a,b vom Typ BOOLEAN):

Verknüpfungen

a AND b	Logisch und
a OR b	Logisch oder
NOT a	Negation von a.

Weiterhin gilt: FALSE < TRUE.

Damit leiten sich ab:

a = b Logische Äquivalenz

a <> b Logisch entweder oder

a <= b Logische Implikation.

Wahrheitstabellen Wahrheitstabellen für die logischen Operatoren:

a	NOT a
TRUE	FALSE
FALSE	TRUE

a	b	a AND b	a OR b	a=b	a<>b	a<=b
TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE

Standardfunktion Außerdem gibt es eine Funktion mit einem Ergebnis vom Typ BOOLEAN:

ODD(i) ergibt TRUE, wenn die ganze Zahl i ungerade ist.

Definition

BOOLEAN

VAR b : BOOLEAN;

Variablen vom Datentyp BOOLEAN sind logische Variablen mit Dateninhalt TRUE oder FALSE.
Logische Operatoren sind: AND, OR, NOT, = , <> , <= .
Eine Standardfunktion mit einem Ergebnis vom Typ BOOLEAN ist ODD(i).

logischer Schalter Hinweis: Ein logischer Schalter ist der Ausdruck:

a := a = FALSE; (mit VAR a:BOOLEAN;)

Wenn a FALSE ist, erhält es den Wert TRUE und umgekehrt.

Logische Ausdrücke und Variablen werden besonders im Zusammenhang mit Schleifen (Kap. 6) und mit Entscheidungen (Kap. 7) gebraucht.

Ein-/Ausgabeformate:

In Standard-Pascal ist die Ausgabe logischer Variablen vorgesehen.

In UCSD-Pascal ist weder Ein- noch Ausgabe logischer Variablen möglich, so daß eine Hilfskonstruktion mit IF...THEN...ELSE benutzt werden muß.

Übungen:

5.5.1 Erfinden Sie eine logische Funktion, die TRUE ergibt, wenn i gerade ist.

5.5.2 Schreiben Sie ein Programm, das prüft, welches von zwei eingegebenen Wörtern das (lexikographisch gesehen) kleinere ist.

5.5.3 Schreiben Sie ein Programm, das eine Wahrheitstabelle für

a => b

erstellt.

Tip: Nach der Pascal-Syntax muß es allerdings a >= b heißen.

5.6 Zusammenfassung

Wir kennen nun die einfachen, d.h. nicht zusammengesetzten Datentypen INTEGER, REAL, CHAR und BOOLEAN sowie den sehr nützlichen Datentyp STRING. Mit diesen Typen können wir schon recht viele Programmierprobleme lösen.

Weiterhin haben wir in diesem Kapitel auch gelernt, welche Operationen und Standardfunktionen mit den dargestellten Datentypen verwendet werden können. Wir kennen die Ausgabeformate und können Typenkonflikte vermeiden.

Eine Hilfe im weiteren Gebrauch ist die Übersichtstabelle der Standardfunktionen mit den zulässigen Datentypen im Anhang.

Immer diese Wiederholungen

Kapitel 6

Bisher haben wir es (mit einigen Ausnahmen) nur mit linearen Algorithmen, d. h. mit Hintereinanderausführungen von Anweisungen zu tun gehabt. Oft jedoch kommt es vor, daß eine oder mehrere Anweisungen mehrfach durchlaufen werden sollen.

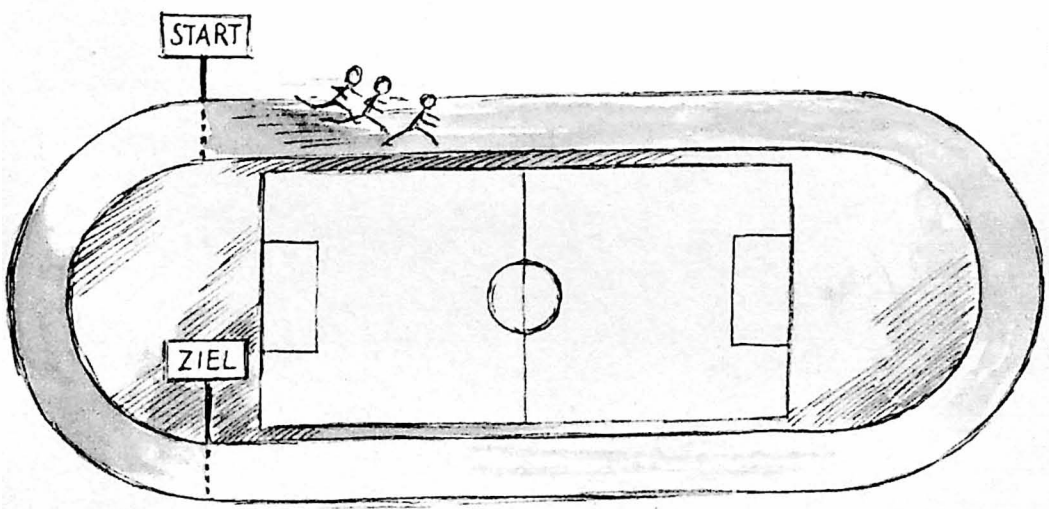
Dazu brauchen wir Strukturen, die es erlauben, sogenannte Schleifen zu programmieren. Dazu gehören Zählschleifen, in denen die Ausführung von Anweisungen mitgezählt und nach einem Endwert beendet wird. Andererseits gibt es Schleifen, deren Abbruch erst durch die Anweisungen in der Schleife selbst bestimmt wird. Im Programm Vielecke in Kap. 3.1 wird z. B. das Zeichnen von Vielecken so lange wiederholt, bis der Benutzer auf die Frage nach weiteren Vielecken mit N antwortet.

Wir behandeln in diesem Kapitel drei Schleifentypen:

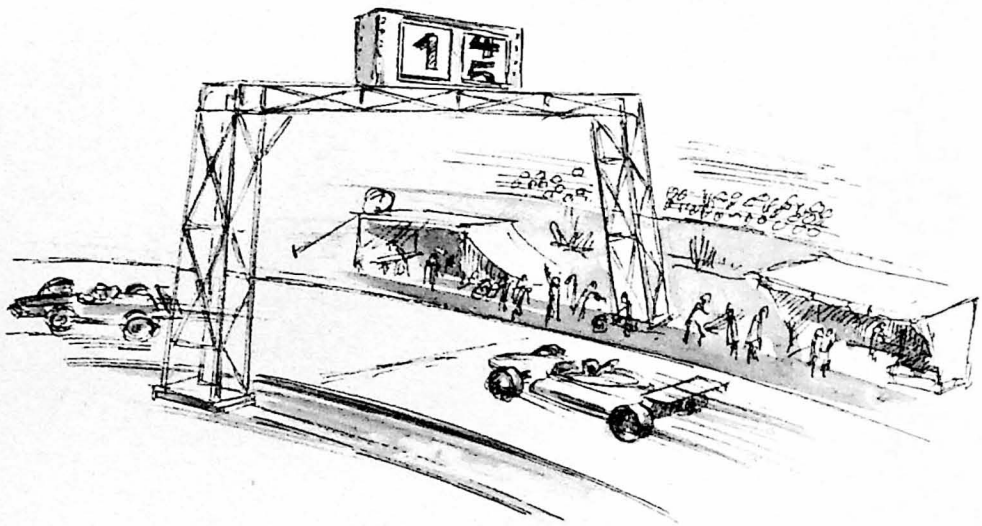
die FOR-, REPEAT- und WHILE-Schleife.

Nach dem Lesen dieses Kapitels

- kennen wir die Syntax der drei Schleifentypen und können sie richtig anwenden;
- wissen wir, welche Unterschiede zwischen den Schleifentypen bestehen;
- kennen wir Fehlerquellen bei der Benutzung von Schleifen und können sie vermeiden;
- kennen wir das Problem der Terminiertheit und können Schleifen korrekt beenden.



6.1 Zähle von ... bis ... – die FOR-Schleife



Rundenzähler

Der Zweck des oben abgebildeten Rundenzählers dürfte bekannt sein: Er zählt Runden. Sehen wir uns seine Arbeitsweise genauer an. Wir lassen dabei die technischen Details außer Betracht. Das Zählwerk des Rundenzählers muß auf einen Anfangswert, z. B. den Wert Null, gesetzt worden sein. Nach jeder Runde wird der aktuelle Wert des Zählwerks um eins erhöht. Dies geschieht so oft, bis das Zählwerk eine vorgegebene Anzahl von Runden anzeigt.

Laufvariable

In Pascal gibt es eine Anweisung, die es ermöglicht, auf ähnliche Art und Weise Runden zu zählen. Diese nennt man FOR-Anweisung. Auch hier wird eine Zählvariable – manchmal sagt man auch Laufvariable – auf einen Anfangswert gesetzt. Ferner muß der Endwert vorgegeben sein.

Den Runden in unserem Beispiel entspricht eine Anweisung oder ein in BEGIN und END gefaßter Anweisungsblock. Die FOR-Schleife hat die Form:

vorwärts

FOR Zählvariable:=Anfangswert TO Endwert DO Anweisung;
FOR Zählvariable:=Anfangswert TO Endwert DO BEGIN Anweisungen END;

rückwärts

Es ist außerdem möglich, rückwärts zu zählen. In diesem Fall muß in der obigen Anweisung TO durch DOWNTO ersetzt werden:

FOR Zählvariable:=Anfangswert DOWNTO Endwert DO Anweisung;
FOR Zählvariable:=Anfangswert DOWNTO Endwert DO BEGIN Anweisungen END;

Die Angabe einer Schrittweite – wie in manchen anderen Programmiersprachen – ist in Pascal nicht möglich. Die Zählvariable wird also immer nur um eins erhöht oder vermindert.

Auflisten des ASCII-Code

Dazu ein kleines *Beispiel:*

In Kap. 5.3 haben wir den ASCII-Zeichensatz kennengelernt. Hier ist nun ein Programm, das alle druckbaren Zeichen dieses Zeichensatzes vorwärts und rückwärts ausgibt.

ASCII

Das Programm:

Programm

```
PROGRAM Ascii;

VAR i,zeit : INTEGER;

BEGIN
  PAGE(OUTPUT);                          (* Loescht den Bildschirm *)
  WRITELN;
  WRITE('Der ASCII-Zeichensatz  ');
  WRITELN(' - vorwaerts:');
  WRITELN;
  FOR i:=32 TO 126 DO
       BEGIN
         WRITE(i:5,':',chr(i):2);
         IF  (i-1)/5 = (i-1) DIV 5  THEN  WRITELN  (* Zeilenvorschub *)
       END;
  FOR zeit:=1 TO 20000 DO;                (* Verzoegerungsschleife *)

  PAGE (OUTPUT);                          (* Loescht den Bildschirm *)
  WRITE('Der ASCII-Zeichensatz  ');
  WRITELN(' - und rueckwaerts:');
  WRITELN;
  FOR i:=126 DOWNTO 32 DO
       BEGIN
         WRITE(i:5,':',chr(i):2);
         IF (i-2)/5 = TRUNC ((i-2)/5) THEN WRITELN  (* Zeilenvorschub *)
       END;
  FOR zeit:=20000 DOWNTO 1 DO;            (* Verzoegerungsschleife *)
  PAGE(OUTPUT)
END.
```

Wenn eine 80-Zeichenkarte vorhanden ist, können die beiden Anweisungen entfallen, die dem Zeilenvorschub dienen (12. und 22. Zeile). Damit kann der Anweisungsblock der beiden Schleifen durch die Anweisung WRITE(i:5,':',chr(i):2); ersetzt werden. Beachtenswert ist der Einsatz von FOR-Anweisungen als Verzögerungsschleife mit einer leeren Anweisung nach dem DO.

Deklariert man die Zählvariable folgendermaßen:

Datentypen der Zählvariablen

```
VAR ch: CHAR;      (* CHAR = Zeichen *)
```

so kann man die entsprechenden FOR-Anweisungen wie folgt ändern:

```
FOR ch:=' ' TO '~' DO      bzw. FOR ch:='~' DOWNTO ' ' DO
```

Die Ausgabe bleibt gleich. Wie an diesem Beispiel zu sehen ist, kann man als Zählvariable auch andere Datentypen als INTEGER, also z. B. CHAR verwenden. Die Datentypen müssen allerdings genau einen Nachfolger oder Vorgänger haben. Daher kommen zunächst nur INTEGER, CHAR und BOOLEAN (nur zwei Werte!) als Zählvariablen in Frage. Später werden wir sehen, wie man selbstdefinierte Typen, sog. Aufzählungstypen, als Zählvariable einsetzen

Zählvariablen

kann. In einem Programm können somit durchaus die folgenden Anweisungen auftreten:

```
FOR ch:='A' TO 'Z' DO WRITE (ch);
(* Gibt alle Grossbuchstaben aus *)

FOR ch:='z' DOWNTO 'a ' DO WRITE (ch);
(*Gibt alle Kleinbuchstaben aus *)
```

Verzögerungsschleife

Wodurch kann nun erreicht werden, daß ein Programm – z.B. mit einer der o.a. FOR-Schleifen – nach der Ausgabe jedes einzelnen Buchstabens kurz anhält und dann fortfährt? Offensichtlich kann eine Verzögerungsschleife innerhalb des Anweisungsblocks einer anderen FOR-Schleife dies bewirken. In einem solchen Fall spricht man von Schachtelung. Dabei ist zu beachten, daß die innere Schleife jeweils vollständig abgearbeitet wird, bevor die äußere Schleife fortgeführt wird.

Aufgabe (Dreieck)

Verdeutlichen wir uns dies anhand eines weiteren Programms. Es soll ein Dreieck aus *-Zeichen bilden und ausgeben. Die Planung des Programms kann zu folgendem Blockdiagramm führen:

Blockdiagramm

Programm DREIECK	
Variablen	Sterne_pro_Zeile, aussen, innen, Anzahl_der_Zeilen: ganze Zahl
Anfang	
Einlesen Anzahl_der_Zeilen	
Setze Sterne_pro_Zeile auf 1	
Zähle Anzahl_der_Zeilen herunter bis 1 und tue	
	Anfang
---→	Gib so viele Leerstellen aus wie noch Zeilen vorhanden
	Zähle von 1 bis Sterne_pro_Zeile und tue
	Gib '*' aus
	Gehe eine Zeile weiter
	Erhöhe die Anzahl der Sterne_pro_Zeile um 2
	Ende
Ende.	

An der mit ---→ gekennzeichneten Stelle fällt auf, daß unsere Planung hier eine unsaubere Formulierung aufweist, d.h. eine Formulierung, die nicht ohne weiteres in die Programmiersprache Pascal zu übersetzen ist. Mit Hilfe des bisher erworbenen Wissens über die FOR-Schleife kann man hier sehr leicht eine Verbesserung vornehmen. Hier nun das zugehörige Programm:

Dreieck

Programm

```
PROGRAM Dreieck;

VAR Sterne_pro_Zeile,
    Aussen, Innen, Leer,
    Anzahl_der_Zeilen: INTEGER;

BEGIN
  PAGE(OUTPUT);
  WRITE ('Wieviel Zeilen soll das Dreieck haben? ');
  READLN (Anzahl_der_Zeilen);
  Sterne_pro_Zeile:=1;            (* fuer erste Zeile *)

  FOR Aussen := Anzahl_der_Zeilen DOWNTO 1 DO
        BEGIN          (* von Aussen *)
          FOR Leer:=Aussen DOWNTO 1 DO WRITE (' '); (* Gibt ' ' aus *)
          FOR Innen:=1 TO Sterne_pro_Zeile DO WRITE ('*');
          Sterne_pro_Zeile := Sterne_pro_Zeile + 2;
          WRITELN        (* naechste Zeile *)
        END            (* von Aussen *)
 END.
```

Schachtelung

Innerhalb der Aussen-Schleife, die einen Anweisungsblock umfaßt, sind zwei weitere FOR-Schleifen untergebracht, die jeweils nur eine Anweisung beinhalten. Ferner wird die Zählvariable Aussen in der Leer-Schleife als Anfangswert eingesetzt. Lassen wir das Programm einmal mit Anzahl_der_Zeilen :=6 ablaufen, so ergibt sich folgender Ausdruck:

Ergebnis

```
     *
    ***
   *****
  *******
 *********
***********
```

Wir halten fest:

Die FOR-Schleife zählt eine Zählvariable von einem Anfangswert bis zu einem Endwert (herauf mit TO, herunter mit DOWNTO).

Regeln

→ Anfangs- und Endwert dürfen Konstanten oder Variablen sein.

→ Anfangs-, Endwert und Zählvariable müssen vom gleichen Datentyp sein.

→ Als Schleifentyp kann jeder Datentyp verwendet werden, der genau einen Nachfolger (bzw. Vorgänger) hat (INTEGER, CHAR, BOOLEAN und Aufzählungstypen).

→ Eine Schleife wird nur einmal abgearbeitet, wenn Anfangs- und Endwert übereinstimmen.

→ Eine Schleife wird keinmal abgearbeitet, wenn bei Verwendung von TO der Anfangswert größer als der Endwert, bei Verwendung von DOWNTO der Anfangswert kleiner als der Endwert ist.

→ Schleifen können geschachtelt werden.

→ Schleifen mit einer leeren Anweisung können als Verzögerungsschleifen eingesetzt werden.

Ändern der Variablen

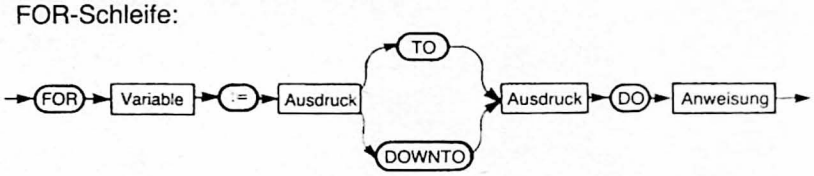
Darüber hinaus

- kann eine Zählvariable zur Berechnung anderer Variablen verwendet werden,
- sollten Zählvariable, Anfangs- und Endwert in einer Schleife nicht verändert werden.

Hinweis: In einigen Pascal-Versionen ist das Ändern von Zählvariablen, Anfangs- und Endwert innerhalb der Schleife zwar möglich, sollte aber aus Gründen eines sauberen Programmierstils unterbleiben.

Definition FOR-Schleife

FOR-Schleife:



FOR Zählvariable:=Anfangswert TO Endwert DO Anweisung(en);
FOR Zählvariable:=Anfangswert DOWNTO Endwert DO Anweisung(en);
sind aufwärtszählende (TO) und abwärtszählende (DOWNTO) Zählschleifen.
Zählvariable, Anfangswert und Endwert sind vom gleichen Datentyp. Sie können von den Typen INTEGER, CHAR, BOOLEAN oder Aufzählungstyp sein.
Die Schleifenvariablen dürfen in der Schleife nicht verändert werden.
Schleifen dürfen geschachtelt werden.

Hinweis: Eine Schleife der Form

```
FOR Zählvariable:=Anfangswert TO Endwert DO;
```

ist ebenfalls möglich. Das Semikolon hinter dem Wort DO stellt eine leere Anweisung dar. Diese Form kann z.B. als Verzögerungsschleife angewendet werden, da der Rechner in dieser Schleife lediglich zählt.

Übungen:

6.1.1 Schreiben Sie ein Programm zu folgender Aufgabe:

a) Eingabe einer positiven ganzen Zahl;
b) die eingegebene Zahl ist (ist nicht) Primzahl.

6.1.2 Fällt Ihnen ein plausibler Grund ein, warum der Endwert einer Schleife innerhalb der Schleife nicht geändert werden soll?

6.1.3 Schreiben Sie ein Programm, das durch mehrfache Multiplikation die ganzzahlige Potenz einer beliebigen Zahl berechnet.

6.1.4 Welche Funktion hat die Schleife

```
FOR i:=5 TO 0 DO ...
```

?

6.1.5 Schreiben Sie ein Grafikprogramm, das eine eckige Spirale zeichnet:

Ausgehend von der Bildschirmmitte, wird der Zeichenstift mit der MOVE-Anweisung um fünf Punkte nach rechts bewegt, dann um 90 Grad gedreht und weiterbewegt. Die Seitenlänge wird jeweils um fünf Einheiten vergrößert. Insgesamt soll die Spirale aus 80 Seiten bestehen.

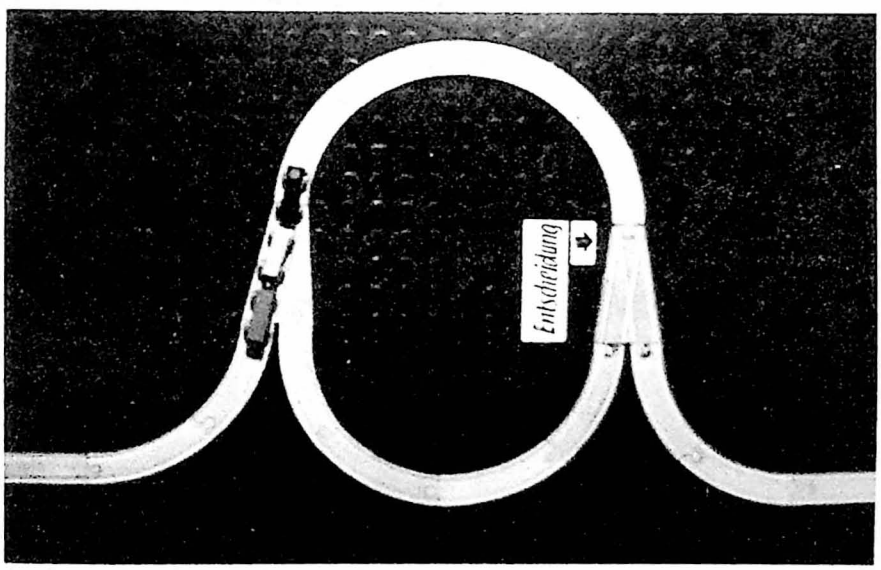
6.1.6 Die Fakultät (in Zeichen: !) von einer ganzen Zahl n ist nichts anderes als die Multiplikation aller Zahlen von 1 bis n.

Beispiel: 5! = 1 * 2 * 3 * 4 * 5

Schreiben Sie ein Programm zur Fakultätsberechnung.

Benutzen Sie den Datentyp Long INTEGER.

6.2 Wiederhole ... bis ... – die REPEAT-Schleife



Abbruchbedingung

Neben der im vorangegangenen Paragraphen besprochenen FOR-Anweisung gibt es in Pascal zwei weitere Möglichkeiten, einen Programmteil wiederholt vom Rechner ausführen zu lassen. Beide unterscheiden sich von der FOR-Anweisung dadurch, daß die Anzahl der Schleifendurchläufe (Wiederholungen) nicht von vorn-

herein durch die Angabe eines Endwertes festgelegt werden muß. Vielmehr wird zur Beendigung der Schleife eine Bedingung – die sogenannte Abbruchbedingung – herangezogen.

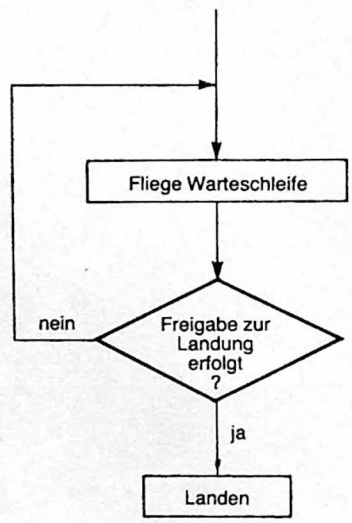
Form Die REPEAT-Schleife ist eine der beiden Möglichkeiten. Sie hat die Form

	REPEAT	Anweisung(en)	UNTIL	Bedingung	
also	WIEDERHOLE	Anweisung(en)	BIS	Bedingung	(erfüllt).

Modell Verdeutlichen wir uns die Arbeitsweise einer solchen Schleife einmal an folgendem Modell: Flugzeuge, die einen Flughafen zur Landung anfliegen, werden gelegentlich angewiesen, eine Warteposition einzunehmen, d.h. Schleifen zu fliegen, bis die Freigabe zur Landung erfolgt. In unserer algorithmischen Sprache muß das Flugzeug die Anweisung

```
WIEDERHOLE Warteschleifen fliegen BIS Freigabe zur Landung
erfolgt
```

Flußdiagramm befolgen. Als Flußdiagramm ergibt sich für unser Modell der REPEAT-Schleife folgendes Bild:



Wenn die Freigabe zur Landung nicht erfolgt, so stürzt unser Flugzeug ab – genau wie unser Programm. Die (Warte-)Schleife wird nur dann verlassen, wenn die Abbruchbedingung erfüllt ist, d.h. den Wert TRUE annimmt.

Sehen wir uns unter diesem Aspekt einmal die folgende Anweisung etwas genauer an:

```
REPEAT WRITE ('Drucker einschalten') UNTIL
Drucker eingeschaltet
```

Ist der Drucker nicht eingeschaltet, so hat unsere Abbruchbedingung, die sich in Pascal leider nicht so einfach darstellen läßt, den Wert FALSE. Dies hat eine Wiederholung der WRITE-Anweisung zur Folge. Erst wenn der Drucker eingeschaltet wird, kann der Rechner die Schleife beenden. Ist kein Drucker vorhanden, so rettet uns nur noch die <RESET>-Taste vor der Wiederholungswut des Rechners.

Terminiertheit

Es ist also peinlich genau darauf zu achten, daß die Abbruchbedingung den Wert TRUE überhaupt annehmen kann (Problem der Terminiertheit). Dies ist natürlich nur dann möglich, wenn die Voraussetzungen für die Bedingung in der Schleife geändert werden.

Ein lauffähiges Beispiel für die Verwendung der REPEAT-Schleife ist das folgende kleine Programm:

Programm

```
PROGRAM Wiederholung;

VAR ch: CHAR;

BEGIN
  REPEAT                                          (* aeussere Schleife *)
    REPEAT                                         (* innere Schleife *)
      PAGE(OUTPUT);      (* loescht den Bildschirm *)
      GOTOXY(3,5);       (* setzt den Cursor auf die 3te Spalte
                            in der 5ten Zeile *)
      WRITE ('Soll die Schleife verlassen werden?  (j/n)');
      READLN (ch)
    UNTIL (ch='j') or (ch='n');                     (* innere Schleife *)
    WRITELN; WRITELN ('Gleich geht es weiter')
  UNTIL ch='j';                                     (* aeussere Schleife *)
  PAGE(OUTPUT);
  GOTOXY(10,15);
  WRITELN ('ENDE')
END.
```

Schachtelung

Zunächst fällt auf, daß REPEAT-Schleifen genau wie FOR-Schleifen geschachtelt werden können.

Was leistet nun das Programm? Wird ein von 'j' oder 'n' verschiedener Buchstabe eingegeben, so wird der Bildschirm gelöscht, der Cursor auf die vorgegebene Stelle gesetzt, die Frage ausgegeben und erneut ein Zeichen eingelesen. Man beachte dabei, daß der Rechner genau zwischen 'j' und 'J' unterscheidet. Die innere Schleife wird also erst verlassen, wenn ein ganz bestimmtes Zeichen (hier: 'j' bzw. 'n') eingelesen wurde. Hat die Variable ch den Wert 'n', so wird die innere Schleife als eine Anweisung der äußeren wiederholt. Andernfalls wird auch die äußere Schleife verlassen und das Programm beendet.

Kein Semikolon vor UNTIL

Bemerkenswert ist ferner, daß die Anweisungen zwischen REPEAT...UNTIL nicht durch BEGIN und END zu einem Anweisungsblock zusammengefaßt sind. Die reservierten Wörter REPEAT...UNTIL übernehmen hier die klammernde Funktion, die sonst BEGIN und END vorbehalten ist. Auch das Semikolon vor UNTIL kann entfallen. Ist es vorhanden, so wird es – wie auch vor END – als leere Anweisung behandelt.

Aufgabe (π)

Wir wollen nun noch ein kleines Problem aus der Mathematik lösen: Berechnung der Kreiszahl π.

Berechnung von π

Dazu benutzen wir das von den Mathematikern bereitgestellte Wissen, daß sich π über eine Produktreihe berechnen läßt:

$$\frac{\pi}{2} = \frac{2^2}{1*3} * \frac{4^2}{3*5} * \frac{6^2}{5*7} * \ldots$$

Allgemein läßt sich der n-te Faktor darstellen als

$$f_n = \frac{4*n^2}{4*n^2-1}$$

Planung Die Zahl π soll mit der größten dem Rechner möglichen Genauigkeit berechnet werden. Dazu brauchen wir eine Schleife, die die oben dargestellten Faktoren so lange miteinander multipliziert, bis sich keine Änderung des Ergebnisses mehr zeigt.

Als Abbruchbedingung könnte man den zuletzt berechneten Wert für π mit dem neuen Wert vergleichen. Durch Rundungsfehler bei REAL-Zahlen (siehe Kap. 5.2) führt dies jedoch selten zu einem Abbruch, denn gleiche Zahlen sind oft für den Rechner nicht gleich. Wir würden damit eine unendlich lange Schleife konstruieren.

Statt dessen benutzen wir eine Hilfsvariable, die den jeweils letzten Wert von η bekommt, subtrahieren die Hilfsvariable vom neuen Wert für η und vergleichen diese Differenz mit einer sehr kleinen Zahl (abhängig von der Rechengenauigkeit des verwendeten Rechners – nicht Null!).

Programm Das Programm:

```
PROGRAM Kreiszahl;

  CONST Delta=1E-9;

  VAR n, Faktor, Pi, Hilf, Differenz : REAL;

  BEGIN
    n := 1;
    Pi := 1;
    REPEAT
      Hilf := Pi;
      Faktor := 4 * SQR(n);
      Pi := Pi * Faktor / (Faktor - 1);
      n := n + 1;
      Differenz := ABS(Hilf - Pi)
    UNTIL Differenz < Delta;
    WRITELN ('Nach ', n:6:1, ' Durchlaeufen ist Pi=', 2*Pi:7:5)
  END.
```

Regeln *Merke:*

→ Die Abbruchbedingung der REPEAT-Schleife muß ein Ausdruck oder eine Variable vom Typ BOOLEAN sein (siehe auch Kap. 5.5).

→ Die Abbruchbedingung muß in der Schleife verändert werden, da die Schleife sonst unendlich lange läuft.

→ Die Schleife wird beendet, wenn die Abbruchbedingung wahr ist.

REPEAT-Schleife

Die besondere Eigenschaft einer REPEAT-Schleife liegt darin, daß die Abbruchbedingung am Ende der Schleife geprüft wird. Daher läuft eine REPEAT-Schleife mindestens einmal.

Eigenschaft der REPEAT-Schleife

Terminiertheit:

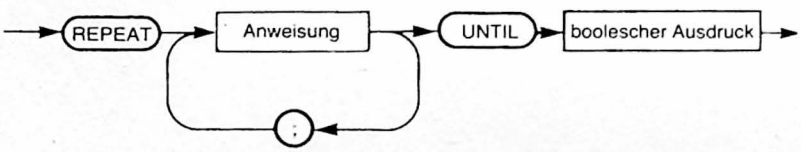
Terminiertheit

Es ist stets genau darauf zu achten, daß die Schleife auch tatsächlich terminiert (beendet) ist. Wir müssen dazu folgendes überprüfen:

- Ist in der Abbruchbedingung überhaupt eine Variable vorhanden, die der Bedingung den Wert TRUE geben kann?
- Wird die Abbruchbedingung jemals erreicht?

Definition REPEAT-Schleife

REPEAT-Schleife:



Der Abbruch der REPEAT-Schleife wird nach Ausführung der Schleifen-Anweisung getestet. Daher läuft die Schleife mindestens einmal.
Die Abbruchbedingung wird in der Schleife verändert.
Auf Terminiertheit der Schleife ist zu achten.

Übungen

6.2.1 Wie kann eine FOR-Schleife durch eine REPEAT-Schleife ersetzt werden?

Lassen Sie auch eine variable Schrittweite zu.

6.2.2 Welches Problem steckt in folgender Schleife?

```
VAR I:INTEGER;

  I:=5;
  REPEAT
     I:=I+1;
  UNTIL I=3;
```

6.2.3 Schreiben Sie ein Programm, das die 2. Wurzel einer beliebigen Zahl berechnet. Verwenden Sie dazu eine Intervallschachtelung, die beendet wird, wenn keine Änderung des berechneten Wertes mehr auftritt.

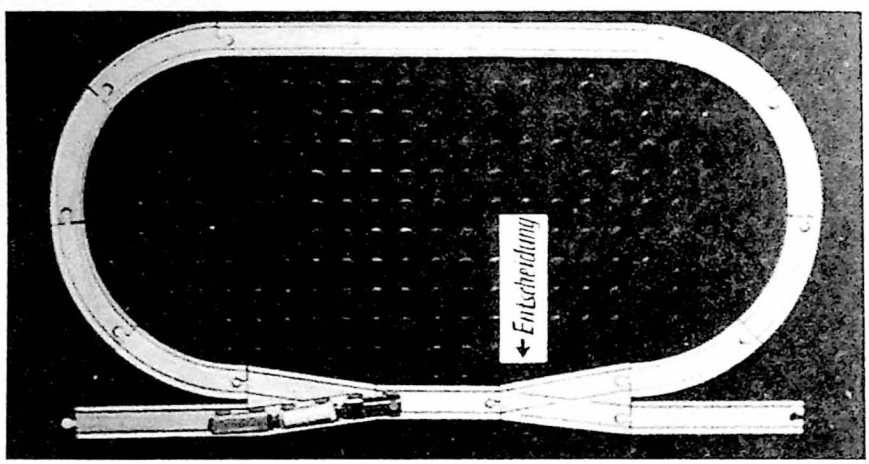
6.2.4 Entwickeln sie ein Programm „Reaktionstest", das bei einem Tastendruck zu zählen beginnt und beim nächsten Tastendruck damit aufhört. Zum Schluß kann der Benutzer wählen, ob er noch einmal spielen möchte.

Verwenden Sie die Funktion KEYPRESS (siehe Kap. 8), die den Wert TRUE bekommt, wenn eine Taste gedrückt wurde.

Als zweite Programmanweisung muß USES APPLESTUFF enthalten sein.

6.2.5 Bestimmen Sie diejenige ganze Zahl N, für die der Wert des Bruchs 1/N eine vorher eingelesene Zahl möglichst genau annähert.

6.3 Solange ... tue ... – die WHILE-Schleife



Modell Zur Veranschaulichung des dritten Typs der in Pascal möglichen Schleifen wollen wir einen Zug benutzen. Dazu lassen wir ihn, wie bei den entsprechenden touristischen Attraktionen üblich, eine ringförmig angelegte Strecke befahren (vgl. obiges Foto). Nur zum Auffüllen von Wasser und Brennstoff und zum Abstellen des Zugs über Nacht wird die Hauptstrecke verlassen und das (einzige) Abstellgleis aufgesucht. Die Betreibergesellschaft einer solchen Museumseisenbahn stellte nun eines Tages einen arg beschränkten, aber trotzdem sehr dienstbeflissenen Lokomotivführer ein. Dieser verließ trotz mannigfaltiger Hinweise und Erläuterungen jedesmal die Hauptstrecke, wenn der Zug an der Weiche zum Abstellgleis angelangt war. Zum Ausgleich stellte er ihn des öfteren nachts auf der Hauptstrecke ab. Daraufhin erhielt der Lokführer folgende „Dienstanweisung", an die er sich peinlich genau hielt und somit alle Unregelmäßigkeiten beseitigte:

```
BEGINNE          (* Tagewerk *)
Zug vom Abstellgleis fahren;

WIEDERHOLE
   SOLANGE noch Kohlen und Wasser vorhanden
   TUE          Strecke befahren;
   Kohlen und Wasser fassen;
BIS es Nacht ist;

Zug auf Abstellgleis fahren
ENDE.            (* Feierabend *)
```

Quersumme

Eingeweihte wissen, daß es sich bei dieser „Dienstanweisung" um die umgangssprachliche Formulierung eines Algorithmus handelt. Untersuchen wir ihn einmal auf seine Struktur. Die uns bereits bekannte Schleife WIEDERHOLE...BIS enthält hier die Anweisung:

```
SOLANGE noch Kohlen und Wasser vorhanden
TUE Strecke befahren;
```

Anweisungen dieses Typs werden wir nunmehr als dritte Möglichkeit der Wiederholung kennenlernen. In unserem Beispiel folgt dem reservierten Wort SOLANGE (engl. WHILE) eine Bedingung. Ist sie wahr, so wird die Anweisung bzw. der in BEGIN und END gefaßte Anweisungsblock ausgeführt, der dem reservierten Wort TUE (engl. DO) folgt. Der Lokomotivführer unserer Museumseisenbahn hat also, *bevor* er die Hauptstrecke befährt, zu prüfen, ob noch genügend Brennstoff und Wasser vorhanden sind. Erst wenn dies der Fall ist, legt unser Zug eine weitere Runde auf der Hauptstrecke zurück. Die Abzweigung zum Abstellgleis wird erst dann benutzt, wenn es an einem von beiden fehlt, d.h. die Bedingung falsch wird.

Zusammengefaßt hat die WHILE-Schleife folgende Form: ***Form***

```
WHILE  Bedingung  DO  Anweisung/-sblock
```

In einem weiteren kleinen Programmspiel wollen wir die Quersumme einer ganzen Zahl berechnen und ausgeben. ***Aufgabe***

Dazu wird eine ganze Zahl eingegeben und folgende Berechnung angestellt:

Zunächst wird die Zahl durch 10 dividiert (mit MOD), und wir addieren den Rest dieser Division in der Variablen Summe auf. Dann wird die Zahl durch 10 dividiert (mit DIV), und wir verwenden diesen ganzzahligen Quotienten für den nächsten Schleifendurchgang. Das Ganze wird so lange wiederholt, wie die Zahl noch größer als Null ist.

Zur Lösung des Problems benutzen wir die WHILE-Schleife. Sie hat die Form: WHILE Bedingung DO Anweisung(en) oder auf deutsch: SOLANGE Bedingung TUE Anweisung(en).

Blockdiagramm

Programm QUERSUMME	
Variablen n,Summe : ganze Zahl	
Anfang	
Eingabe der Zahl n	
Summe := 0	
Solange n > 0 tue	
	Summe := Summe + n MOD 10
	n := n DIV 10
Ausgabe der Quersumme der Zahl	
Ende.	

Regeln

Das Programm:

Programm

```
PROGRAM Quersumme;

  VAR n, Summe : INTEGER;

  BEGIN
    WRITE ('Eingabe der Zahl: ');
    READLN (n);
    Summe := 0;
    WHILE n > 0 DO BEGIN
      Summe := Summe + n MOD 10;
      n := n DIV 10
    END;  (* der WHILE-Schleife *)
    WRITELN ('Die Quersumme ist: ', Summe:4)
  END.
```

Regeln Solange die Bedingung den Wahrheitswert TRUE hat, läuft die Schleife.

Merke:

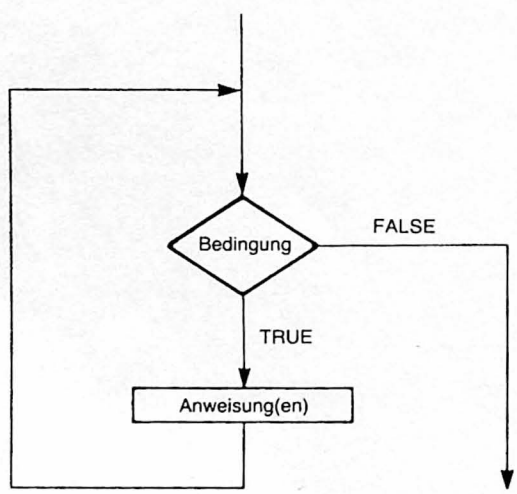
→ Die Bedingung der WHILE-Schleife muß ein Ausdruck oder eine Variable vom Typ BOOLEAN sein (siehe auch Kap. 5.5).

→ Die Bedingung muß in der Schleife verändert werden, da die Schleife sonst unendlich lange läuft.

→ Die Schleife wird beendet, wenn die Bedingung falsch ist.

WHILE-Schleife Die besondere Eigenschaft einer WHILE-Schleife liegt darin, daß die Bedingung am Anfang der Schleife geprüft wird. Daher läuft eine WHILE-Schleife möglicherweise keinmal.

Flußdiagramm In einem Flußdiagramm läßt sich das wie folgt darstellen:



Terminiertheit Terminiertheit:

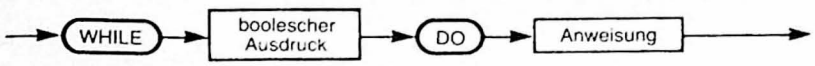
Es ist peinlich genau darauf zu achten, daß die Schleife auch tatsächlich terminiert (beendet) ist. Wir müssen dazu folgendes überprüfen:

WHILE-Schleife

– Ist in der Bedingung überhaupt eine Variable vorhanden, die der Bedingung den Wert TRUE geben kann?
– Wird die Bedingung jemals erreicht?

Definition WHILE-Schleife

WHILE-Schleife:



Der Abbruch der WHILE-Schleife wird vor Ausführung der Schleifen-Anweisungen getestet. Daher läuft die Schleife möglicherweise keinmal.
Die Bedingung wird in der Schleife verändert.
Auf Terminiertheit der Schleife ist zu achten.

REPEAT – WHILE

Unterschied REPEAT – WHILE:

An dieser Stelle sei noch einmal auf die Unterschiede der Schleifenarten hingewiesen:

Bei der REPEAT-Schleife wird die Bedingung nach Ausführung der Schleifenanweisungen auf den Wahrheitswert TRUE überprüft, bei der WHILE-Schleife vorher.

Die REPEAT-Schleife läuft, während die Bedingung den Wert FALSE hat (d.h. bis sie TRUE ist) – die WHILE-Schleife läuft, solange die Bedingung den Wert TRUE hat.

Ersetzung

Man kann beide Schleifenformen durch die jeweils andere ersetzen.

Dazu unser Programm QUERSUMME mit einer REPEAT-Schleife:

Programm

```
PROGRAM Quersumme;

  VAR n, Summe : INTEGER;

  BEGIN
    WRITE ('Eingabe der Zahl: ');
    READLN (n);
    Summe := 0;
    REPEAT
      Summe := Summe + n MOD 10;
      n := n DIV 10
    UNTIL n <= 0;
    WRITELN ('Die Quersumme ist: ', Summe:4)
  END.
```

Übungen:

6.3.1 Ersetzen Sie folgende REPEAT-Schleife durch eine gleichwertige WHILE-Schleife:

```
VAR I:INTEGER;
    ch:CHAR;
    ...
    I:=0;
    REPEAT
       I:=I+2;
       WRITELN (I:4);
       WRITE ('Noch eine Berechnung (J/N)? ');
       READ (ch)
    UNTIL ch='N';
    ...
```

6.3.2 Ersetzen Sie die REPEAT-Schleife im Programm Kreiszahl aus Kap. 6.2 durch eine gleichwertige WHILE-Schleife.

6.3.3 Schreiben Sie ein Programm „Spuren", das so lange weiße Strecken zu jeweils weiteren, zufällig berechneten Koordinaten zeichnet, bis eine Taste gedrückt wird.

Tip:

1. Zweite Programmzeile: USES APPLESTUFF,TURTLEGRAPHICS;
2. KEYPRESS ist TRUE, wenn eine Taste gedrückt wurde.
3. RANDOM ist eine zufällige INTEGER-Zahl (achten Sie auf die Grenzen des Koordinatensystems!).
   (Siehe Kap. 8)

6.4 Zusammenfassung

In diesem Kapitel haben wir drei Schleifenformen kennengelernt. Dies ermöglicht es uns, die Fähigkeiten des Computers (schnelle und oft wiederholte Ausführung von Anweisungen) noch besser als bisher auszunutzen.

Wir können eine Schleife benutzen, die eine bestimmte Anzahl von Durchgängen wiederholt (FOR) oder zwei Schleifen (REPEAT und WHILE), die durch Bedingungen (boolesche Ausdrücke) gesteuert werden. Bei letzteren haben wir sehr genau darauf zu achten, daß die Schleifen auch beendet (terminiert) sind.

REPEAT und WHILE lassen sich zwar gegeneinander austauschen, sollten aber im Sinne einer übersichtlichen und algorithmisch eleganten Programmierung beide gebraucht werden.

Entscheidungen

Kapitel 7

Schon bei der Behandlung der Schleifen (Kap. 6) hatten wir es nicht mehr mit linearen Algorithmen und Programmen zu tun, in denen alle Anweisungen hintereinander ausgeführt werden. Je nach Bedingung konnte eine Anzahl von Anweisungen wiederholt werden.

In diesem Kapitel werden wir die Hintereinanderausführung von Befehlen noch weiter durchbrechen. Eine oder mehrere Entscheidungen werden zu einer Verzweigung des Programms führen.

Nach dem Lesen dieses Kapitels

- können wir die IF...THEN-Anweisung für eine einfache Verzweigungsmöglichkeit anwenden;
- können wir die IF...THEN...ELSE-Anweisung für eine zweifache Verzweigungsmöglichkeit anwenden;
- können wir die CASE-Anweisung für mehrfache Verzweigungsmöglichkeiten anwenden;
- können wir Verzweigungen verschachteln;
- erkennen wir Fehler in Verzweigungen und können sie vermeiden.

7.1 Wenn ... dann ... sonst ... – die IF...THEN...ELSE-Konstruktion

Fragen eines Computerfans

Entscheidungen gibt's ...

> Soll ich mir einen Computer kaufen oder nicht?
>
> Diese Frage kann ich mir ganz einfach beantworten:
>
> Wenn
> ich genug Geld habe UND einen Computer besitzen möchte,
> dann
> gehe ich in einen Computer-Shop und kaufe einen.
>
> Was ist aber, wenn die Bedingungen nicht zutreffen?
>
> Nun, dann geht das Leben eben weiter.

Wahrheitswerte

In diesem Monolog (den manch ein Computerfan kennt) kommt eine Entscheidung vor, die von zwei Bedingungen abhängt. Beide Bedingungen müssen zutreffen, was durch das Wort UND bestimmt wird. Wir kennen schon Ausdrücke und Verknüpfungen dieser Art. Sie sind vom Typ BOOLEAN, denn sie können die Wahrheitswerte wahr oder falsch annehmen. Wenn also beide Bedingungen wahr sind, dann habe ich grünes Licht für den Computerkauf. Sollte das nicht der Fall sein, so geht mein Leben programmgemäß weiter.

Bedingungen

Auch in der Programmiersprache Pascal gibt es diese Situation. Oft sollen eine oder mehrere Anweisungen nur dann ausgeführt werden, wenn eine Bedingung oder eine Kombination von Bedingungen erfüllt ist. Dazu übersetzen wir einfach „wenn...dann" ins Englische und erhalten IF...THEN. Dieses sind genau die reservierten Wörter für eine bedingte Ausführung von Anweisungen.

IF...THEN

Die Form ist:

```
IF logischer Ausdruck THEN Anweisung(en);
```

logische Ausdrücke

Wenn mehrere Anweisungen ausgeführt werden sollen, so werden sie wieder mit BEGIN...END zusammengefaßt. Der logische Ausdruck muß den Wahrheitswert TRUE haben, damit die Anweisung(en) ausgeführt wird (werden), andernfalls wird das Programm weiter fortgeführt.

Aus Kap. 5.5 kennen wir schon logische Ausdrücke. Beispiele sind:

```
B                : wobei B vom Datentyp BOOLEAN ist
B=TRUE
B=FALSE
X < Y            : wobei X und Y vom gleichen Typ sind
(X < Y) AND (X > Z)
NOT (A = B)
```

Hinweis: Wenn mehrere logische Ausdrücke durch Verknüpfungsoperatoren (AND, OR, NOT) zusammengefaßt werden, so ist auf korrekte Klammerung zu achten!

bedingter Zeilenvorschub

Wir wollen uns nun ein kleines Beispiel anschauen:

Programm

```
PROGRAM Waswohl;

  VAR I : INTEGER;

  BEGIN
    FOR I:=1 TO 99 DO BEGIN
      WRITE (I:3);
      IF I MOD 9 = 0 THEN WRITELN
    END (* der FOR-Schleife *)
  END.
```

Frage: Was bewirkt dieses kleine Programm?

Ergebnis

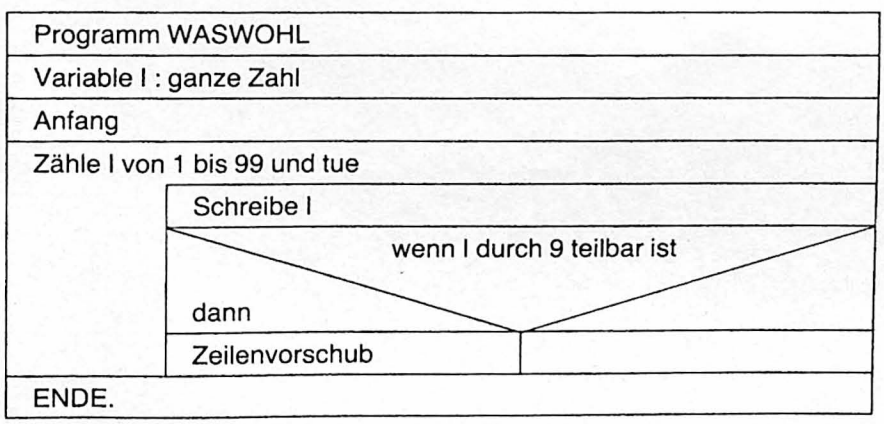
Einerseits könnten wir es in den Rechner tippen und ausprobieren, andererseits läßt sich die Funktion natürlich auch auf dem Papier ermitteln:

Im Programm läuft eine Schleife von 1 bis 99. Bei jedem Schleifendurchlauf wird die Schleifenvariable ohne Zeilenvorschub in einem Feld von drei Zeichen geschrieben. Wenn sich allerdings die Variable durch 9 teilen läßt (I MOD 9 = 0), dann wird ein Zeilenvorschub gemacht, so daß die Zahlen in Neunerkolonnen geschrieben werden:

```
  1   2   3   4   5   6   7   8   9
 10  11  12  13  14  15  16  17  18
 19  20  21  22  23  24  25  26  27
 28  29  30  31  32  33  34  35  36
 37  38  39  40  41  42  43  44  45
 46  47  48  49  50  51  52  53  54
 55  56  57  58  59  60  61  62  63
 64  65  66  67  68  69  70  71  72
 73  74  75  76  77  78  79  80  81
 82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99
```

Im Blockdiagramm sieht das Programm folgendermaßen aus:

Blockdiagramm



zweiseitige Entscheidungen

In dem vorangegangenen Problem handelte es sich um eine Entscheidung mit nur einer Verzweigungsmöglichkeit, denn für den Fall, daß die Bedingung das Ergebnis FALSE hat, wurde mit der nächsten Anweisung fortgefahren.

zweiseitige Entscheidung

Nun sind in Pascal aber auch Entscheidungen mit zwei Verzweigungsmöglichkeiten vorgesehen. Das Wörtchen ELSE (andernfalls) gibt uns die Möglichkeit, bei negativem Ausgang (FALSE) der Entscheidung einen anderen Anweisungsteil ausführen zu lassen.

IF...THEN...ELSE

Die Form ist:

```
IF logischer Ausdruck THEN Anweisung(en) ELSE Anweisung(en);
```

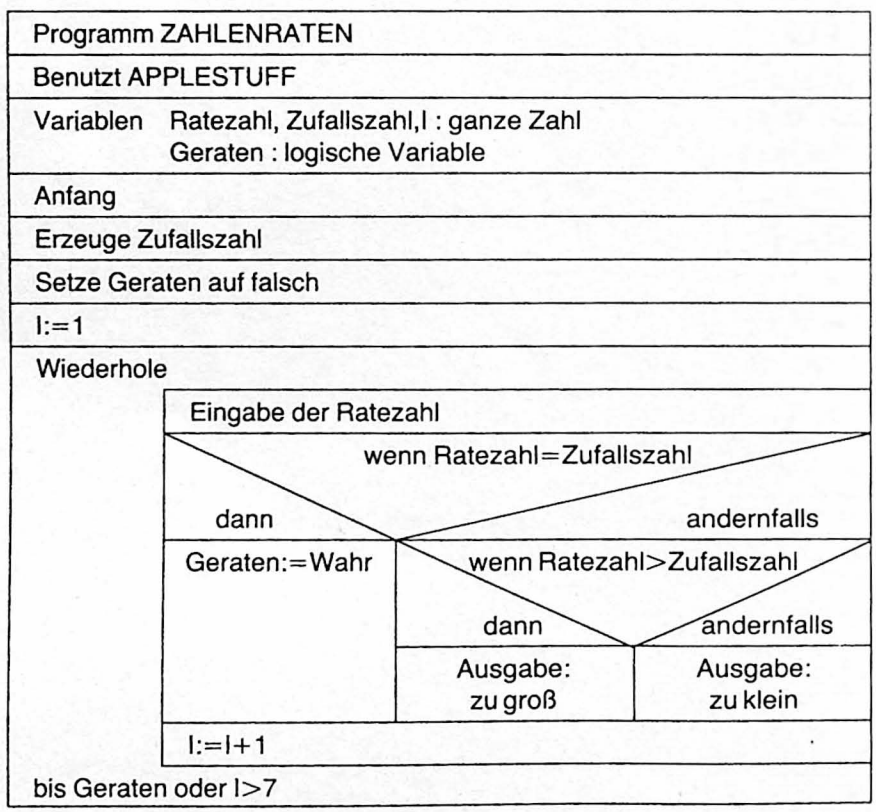
Achtung: Vor ELSE darf kein Semikolon stehen!

Aufgabe (Zahlenraten)

Wir wollen nun ein Beispielprogramm entwerfen: Zahlenraten. Der Rechner erzeugt eine Zufallszahl zwischen 0 und 99.Der Benutzer darf siebenmal eine Zahl raten. Wenn die Zahl größer oder kleiner ist als die Zufallszahl, dann wird dies dem Benutzer mitgeteilt. Bei richtiger Eingabe der Zahl beglückwünscht der Rechner den Benutzer.

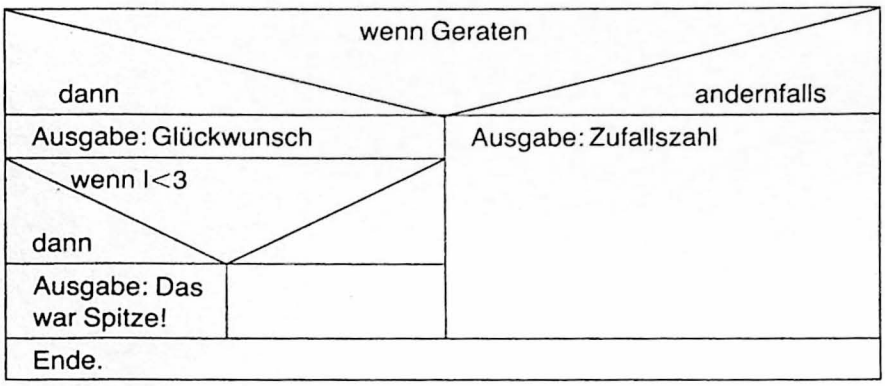
Bemerkung: Die Erzeugung von Zufallszahlen zwischen 0 und MAXINT geschieht beim APPLE durch die Funktion RANDOM in der Bibliothek APPLESTUFF. Wir müssen also im Programmkopf vermerken: USES APPLESTUFF;. Soll bei jedem neuen Durchlauf des Programms eine neue Zufallszahl erzeugt werden, so geben wir RANDOMIZE ein.

Blockdiagramm



Zahlenraten

(Fortsetzung des Diagramms)



Das Programm:

Programm

```
PROGRAM Zahlenraten;
  USES APPLESTUFF; (* fuer Zufallszahlen *)

  VAR Ratezahl, Zufallszahl, I : INTEGER;
      Geraten : BOOLEAN;

  BEGIN
    WRITELN ('Zahlenraten zwischen 0 und 99');
    WRITELN;
    RANDOMIZE; (* erzeugt neue Zufallszahl *)
    Geraten := FALSE;
    Zufallszahl := RANDOM MOD 100;  (* Zahl zwischen 0 und 99 *)
    I := 1;
    REPEAT
      WRITE ('Ratezahl: ');
      READLN (Ratezahl);
      IF Ratezahl=Zufallszahl THEN Geraten:=TRUE
      ELSE BEGIN
        IF Ratezahl>Zufallszahl THEN WRITELN ('zu gross')
        ELSE WRITELN ('zu klein')
      END; (* von ELSE *)
      I:=I+1
    UNTIL Geraten OR (I>7);
    IF Geraten THEN BEGIN
      WRITELN ('Herzlichen Glueckwunsch!');
      IF I<3 THEN WRITELN ('Das war Spitze!')
    END (* von IF *)
    ELSE WRITELN ('Die Zahl war: ',Zufallszahl:3)
  END.
```

Kleine Aufgabe: Verändern Sie das Programm so, daß der Benutzer entscheiden kann, ob er das Spiel noch einmal spielen will.

Schachtelung

In dem vorangegangenen Programm wurden auch geschachtelte Entscheidungen verwendet. Dies ist selbstverständlich bei beiden IF-Konstruktionen möglich. Dabei ist zu beachten, daß eine IF-Abfrage, die von einer anderen abhängt, nur dann ausgeführt wird, wenn die erste Abfrage ein wahres Ergebnis hat.

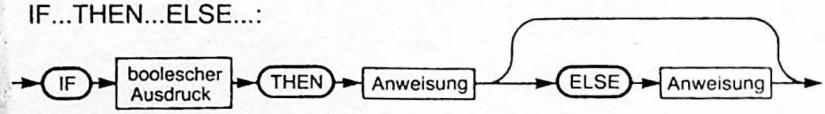
Beispiel:

```
IF Tag=13 THEN IF Wochentag=Freitag THEN WRITELN
('Vorsicht heute!');

IF (Tag=13) AND (Wochentag=Freitag) THEN WRITELN
('Vorsicht heute!');
```

Beide Konstruktionen haben den gleichen Effekt. Nur wenn die Variable Tag gleich 13 ist *und* die Variable Wochentag gleich Freitag ist, wird der Text geschrieben. Die erste (geschachtelte) Konstruktion hat aber den Vorteil, daß *nur*, wenn Tag gleich 13 ist, der Wochentag auch noch überprüft wird. Dies spart natürlich Rechenzeit, da in der zweiten Konstruktion beide Variablen immer geprüft werden.

Definition IF...THEN...ELSE

IF...THEN...ELSE...:



Die Konstruktion
IF logischer Ausdruck THEN Anweisung(en)
ELSE Anweisung(en)
stellt eine zweifache Verzweigung dar. Wenn der logische Ausdruck wahr ist, wird der 1. Anweisungsteil ausgeführt, andernfalls der 2. Anweisungsteil.

Hinweis: In dem Syntaxdiagramm für IF...THEN...ELSE steckt auch die IF...THEN-Konstruktion. Dies ist das allgemeinere Diagramm.

Fehlerquellen

Fehlerquellen: Bei geschachtelten Verzweigungen kann es insbesondere mit der IF...THEN...ELSE-Konstruktion leicht zu Fehlern kommen, wenn man nicht sorgfältig plant.

Schachtelung

Hier ein Beispiel einer Schachtelung:

```
IF Spannung >= 2 THEN
   IF Spannung > 20 THEN
      IF Spannung > 100 THEN
         WRITELN ('Bereich ueberschritten')
      ELSE WRITELN ('großer Bereich')
   ELSE WRITELN ('normaler Bereich')
ELSE WRITELN ('Spannung unter 2 V');
```

Diese Meßbereichsauswahl für ein Spannungsmeßgerät hat folgende Funktion:

bei Spannungen größer als 100 V Meßbereichsüberschreitung;
bei Spannungen zwischen 20 V und 100 V großer Bereich;
bei Spannungen zwischen 2 V und 20 V normaler Bereich;
bei Spannungen unter 2 V Meßbereich unter 2 V.

Übungen:

7.1.1 Schreiben Sie ein Programm, das nach Eingabe der Koeffizienten eine quadratische Gleichung löst (Tip: Formelsammlung Mathematik). Bedenken Sie auch die Fälle allgemeingültiger und unlösbarer Gleichungen.

7.1.2 Ein Programm soll nach Eingabe einer Stundenzahl den Wochenarbeitslohn berechnen. Stunden, die über 40 Stunden

hinausgehen, sind Überstunden. Die Stundensätze für normale Arbeitsstunden und Überstunden werden als Konstanten gesetzt.

7.1.3 Das Maximum von vier eingegebenen Zahlen soll bestimmt und ausgegeben werden.

7.1.4 Schreiben Sie ein Programm, das alle Primzahlen innerhalb zweier eingegebener Grenzen bestimmt und ausgibt.

7.1.5 Ein Programm soll ermitteln, ob eine eingegebene Zahlenfolge (Ende bei 9999) steigend, fallend oder unsortiert ist.

7.1.6 Schreiben Sie für ein kleines Kind auf, wie es sich an einer Fußgängerampel zu verhalten hat.

7.2 Case für Mehrfachentscheidungen und Schreibfaule

Programm

```
PROGRAM Bruchrechnung;

  VAR Z1, Z2, Z3 : INTEGER;
      N1, N2, N3 : INTEGER;
      I, Min, Teiler : INTEGER;
      Operator : CHAR;

  BEGIN
    WRITELN ('Eingabe des 1. Bruches: ');
    WRITE ('Zaehler: ');
    READLN (Z1);
    WRITE ('Nenner:  ');
    READLN (N1);
    WRITELN ('Eingabe des 2. Bruches: ');
    WRITE ('Zaehler: ');
    READLN (Z2);
    WRITE ('Nenner:  ');
    READLN (N2);
    WRITE ('Operator (+,-,*,/): ');
    READ (Operator);

    CASE Operator OF
      '+' : BEGIN
              Z3 := Z1 * N2 + Z2 * N1;
              N3 := N1 * N2
            END;
      '-' : BEGIN
              Z3 := Z1 * N2 - Z2 * N1;
              N3 := N1 * N2
            END;
      '*' : BEGIN
              Z3 := Z1 * Z2;
              N3 := N1 * N2
            END;
      '/' : BEGIN
              Z3 := Z1 * N2;
              N3 := Z2 * N1
            END
    END; (* CASE *)

    IF Z3 < N3 THEN Min:=Z3
               ELSE Min:=N3;
    FOR I:=1 TO Min DO BEGIN
      IF Z3 MOD I = 0 THEN
        IF N3 MOD I = 0 THEN
          Teiler := I
    END; (* FOR *)

    Z3 := Z3 DIV Teiler;
    N3 := N3 DIV Teiler;
    WRITELN;
```

(Fortsetzung des Programms)

```
    WRITELN ( Z1:4,'    ',Z2:4,'    ',Z3:4 ):
    WRITELN ( '---- ',Operator,' ---- = ---- ' ):
    WRITELN ( N1:4,'    ',N2:4,'    ',N3:4 )
END.
```

Fallunterscheidung

Wir haben es hier mit einer Fallunterscheidung zu tun. Für den Fall (CASE), daß Operator von (OF) der Form einer der folgenden Fälle ist, wird ein Anweisungsteil ausgeführt. Die ganze Fallunterscheidung wird mit END abgeschlossen (vor dem END braucht kein Semikolon zu stehen).

Aufgabe (Bruchrechnung)

Das Programm führt eine Bruchrechnung aus.

Der Benutzer gibt zwei Brüche mit Zähler und Nenner ein sowie einen Operator (+, −, ∗, /). Das Programm rechnet das Ergebnis aus und gibt dieses gekürzt aus.

Folgende Stichworte wissen wir noch aus der Mathematik der Brüche:

Addition: Bringe auf Hauptnenner und addiere Zähler.

Subtraktion: Bringe auf Hauptnenner und subtrahiere Zähler.

Multiplikation: Multipliziere jeweils Zähler und Nenner.

Division: Multipliziere mit Kehrwert.

Einfachste Methode für den Hauptnenner: Multiplikation der Nenner.

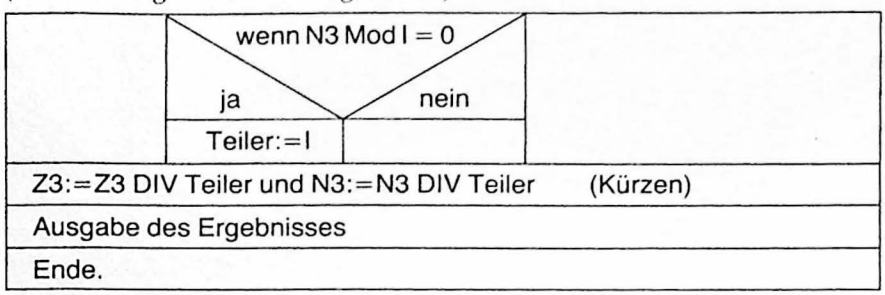
Das Blockdiagramm dazu:

Blockdiagramm

<table>
<tr><td colspan="4">Programm BRUCHRECHNUNG</td></tr>
<tr><td colspan="4">Variablen Z1, Z2, Z3 : ganze Zahl (Zähler)
N1, N2, N3 : ganze Zahl (Nenner)
I, Min, Teiler : ganze Zahl
Operator : Zeichen</td></tr>
<tr><td colspan="4">Anfang</td></tr>
<tr><td colspan="4">Einlesen der Brüche mit Z1, N1, Z2, N2 und des Operators</td></tr>
<tr><td colspan="4">falls Operator ist</td></tr>
<tr><td>+</td><td>−</td><td>∗</td><td>/</td></tr>
<tr><td>Z3=Z1∗N2+Z2∗N1
N3=N1∗N2</td><td>Z3=Z1∗N2−Z2∗N1
N3=N1∗N2</td><td>Z3=Z1∗Z2
N3=N1∗N2</td><td>Z3=Z1∗N2
N3=N1∗Z2</td></tr>
<tr><td colspan="4">wenn Z3 < N3
ja nein</td></tr>
<tr><td colspan="2">Min:=Z3</td><td colspan="2">Min:=N3</td></tr>
<tr><td colspan="4">zähle I von 1 bis Min und tue
wenn Z3 MOD I = 0
ja nein</td></tr>
</table>

Menü

(Fortsetzung des Blockdiagramms)



Funktion

Das Kürzen geht folgendermaßen vor sich: Der Teiler, durch den später Zähler und Nenner gekürzt werden, kann höchstens so groß sein wie die kleinere der beiden Zahlen (Min). Dann wird eine Schleife durchlaufen, in der versucht wird, Zähler *und* Nenner durch die Laufvariable I zu teilen. Für I=1 funktioniert dieses immer. Wenn sich allerdings noch ein größeres I findet, so wird der Bruch echt gekürzt.

Ergebnis

Beispiel für die Ausgabe des Programms:

```
Eingabe des 1. Bruches:
Zaehler:  1
Nenner:  15
Eingabe des 2. Bruches:
Zaehler:  5
Nenner:  6
Operator (+,-,*,/): +

  1     5     9
--- + --- = ---
 15     6     0
```

mehrere Konstanten

Vor den Doppelpunkten der CASE-Anweisung können auch mehrere Werte stehen, die dann mit Kommata getrennt werden. Beispiel:

```
CASE Frage OF
   'E','e' : Eingabe;
   'A','a' : Ausgabe;
   'S','s' : Sortieren;
   'D','d' : Drucken;
   'F','f' : Finden;
   'Z','z' : EXIT (PROGRAM)
END;
```

Eine solche CASE-Anweisung könnte einem Menü entstammen. Hierbei wird dem Benutzer z. B. auf dem Bildschirm angeboten:

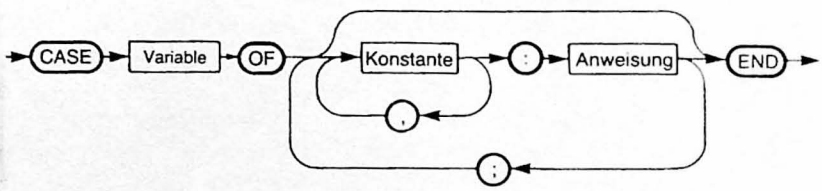
Menü

```
Datenverarbeitung, waehlen Sie:
   E(ingabe von Daten
   A(usgabe von Daten
   S(ortieren
   D(rucken
   F(inden nach Kriterien
   Z(um Schluss
```

Der Benutzer braucht nur den ersten Buchstaben (Eingabe z. B. mit READ oder READ(KEYBOARD,..)) seiner Wahl einzutippen, und das Programm führt entsprechende Anweisungen (evtl. ganze Prozeduren) aus.

Definition CASE

CASE:



Die Mehrfachentscheidung mit CASE hat die Form:

```
CASE Variable OF
    Wert1 : Anweisung(en)1;
    Wert2 : Anweisung(en)2;
    Wert3 : Anweisung(en)3;
    ....
    Wertn : Anweisung(en)n
END;
```

Wenn ein Anweisungsteil von mehreren Werten abhängen soll, so werden die Werte durch Kommata getrennt.
Die Variable muß von einem skalaren Datentyp, nicht aber vom Typ REAL sein, d.h. Aufzählungstyp, Unterbereich, INTEGER, CHAR, BOOLEAN.

Fehlerquellen

Fehlerquellen: Wenn die Variable keinen der unter CASE genannten Werte annimmt, so wird eine Leeranweisung ausgeführt. Im UCSD- und Standard-Pascal ist kein ELSE vorgesehen, das für diesen Fall gebraucht werden könnte!

Es sind nur skalare Datentypen als Variablen zugelassen. Als bekannte Typen können wir INTEGER, CHAR und BOOLEAN benutzen. Später werden wir noch Aufzählungstypen und Unterbereiche kennenlernen, die auch für CASE zugelassen sind.

Übungen:

7.2.1 Sie kennen aus Ihrer bisherigen Arbeit mit dem Computer schon die Menüsteuerung. Woher?

7.2.2 Schreiben Sie ein Programm „Quiz", bei dem (etwa 5) Fragen gestellt werden und der Benutzer jeweils eine von vier vorgeschlagenen Antworten geben muß (dazu wird ein Buchstabe eingegeben).

7.2.3 Wie läßt sich die folgende Mehrfachentscheidung durch eine mehrstufige Entscheidung mit IF...THEN...ELSE ersetzen?

```
CASE i OF
   1 : Teil1;
   2 : Teil2;
   3 : Teil3;
   4 : Teil4;
   5,6 : Teil5
END;
```

7.3 Zusammenfassung

Nach den Schleifen haben wir nun sogenannte Verzweigungen kennengelernt, mit deren Hilfe wir aus einem starr geradeaus verlaufenden Algorithmus (Programm) ausbrechen können.

Genaugenommen sind es zwei Verzweigungsarten, die hier vorgestellt wurden: IF...THEN...ELSE und CASE. Die IF...THEN-Konstruktion ist ja in der IF...THEN...ELSE-Konstruktion enthalten, wenn der ELSE-Teil weggelassen wird.

In der Programmiersprache BASIC ist mit diesen Konstruktionen oft die Möglichkeit eines Sprungs zu einer anderen Programmstelle verbunden. In Pascal kommen wir ohne die Sprünge aus. Hier hat die Verzweigung den Sinn, daß, je nach Bedingung, eine Anweisung (oder ein Block von Anweisungen) ausgeführt wird. Die Anweisung kann auch ein Prozedurname sein, hinter dem sich ein größerer Programmteil verbirgt.

Mit der CASE-Anweisung haben wir eine sehr komfortable Möglichkeit, viele verschiedene Bedingungen abzufragen und auszuwerten. Leider ist nur in sehr wenigen Pascal-Implementierungen eine ELSE-Anweisung in Verbindung mit der CASE-Anweisung anzutreffen. Diesen Mangel heißt es, mit Programmiergeschick zu umgehen.

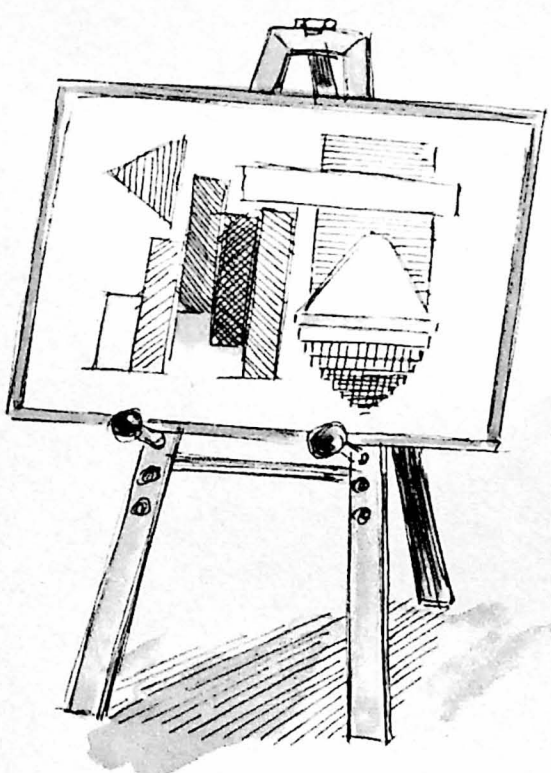
Kleiner Ausflug in die Grafik

Kapitel 8

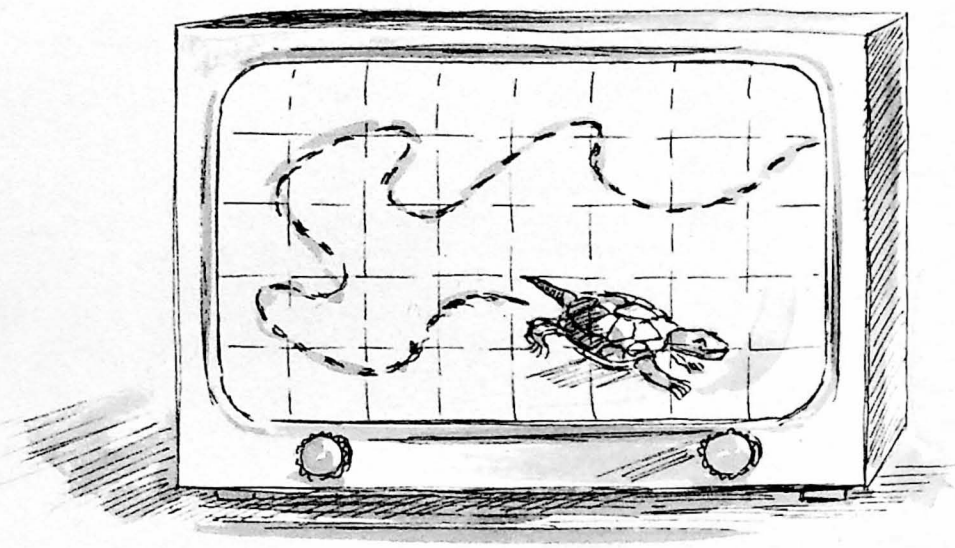
In diesem Kapitel wollen wir uns mit den Möglichkeiten grafikfähiger Rechner im UCSD-Pascal beschäftigen. Alle Programmbeispiele sind direkt auf einem APPLE II lauffähig. Bei anderen Rechnern mit der sogenannten Turtlegrafik muß eventuell ein anderer Bildschirmaufbau beachtet werden und müssen die entsprechenden Längen umgerechnet werden.

Nach dem Lesen dieses Kapitels

- können wir die Befehle der Turtlegrafik richtig anwenden,
- wissen wir, was eine LIBRARY ist,
- kennen wir die APPLE-LIBRARY,
- haben wir Spaß an der Grafik gehabt.



8.1 Was hat die Schildkröte mit dem Computer zu tun?



Farbspur

Wir haben in unserem Rechner eine recht intelligente und künstlerisch begabte Schildkröte versteckt. Sie kann von unseren Programmen aus bewegt werden und dabei eine Spur in einer Farbe hinterlassen. Dazu haben wir Anweisungen, die der Schildkröte sagen, in welcher Farbe die Spur sein soll, wie weit sie sich bewegen soll, wohin sie sich drehen und wohin sie sich bewegen soll. Außerdem können wir den Bewegungsspielraum durch geeignete Befehle einengen oder ausweiten.

Turtle

Im Englischen heißt die Schildkröte Turtle. Im englischsprachigen Raum ist – insbesondere für den Anfangs-Mathematikunterricht – eine Geometrie bekannt, die auf Bewegungen dieser Schildkröte aufbaut und sich Turtlegrafics nennt. Voraussetzung zur Ausführung dieser Grafik ist die Möglichkeit des Rechners zur Darstellung der Grafik auf dem Bildschirm. Beim APPLE z. B. ist diese Möglichkeit vorhanden. Die Anweisungen zur Ausführung der Grafik sind in

UNIT

einer UNIT in der Bibliothek (SYSTEM.LIBRARY) enthalten. Eine UNIT ist ein Programm, das Prozeduren und Funktionen enthält, die durch Nennung des Prozedur- oder Funktionsnamens aufgerufen werden und benutzt werden können wie Pascal-Befehle. Wenn die entsprechende UNIT vom Programm benutzt wird, so können die Prozeduren und Funktionen verwendet werden, als wenn sie im Programm geschrieben worden sind.

Turtle-graphics

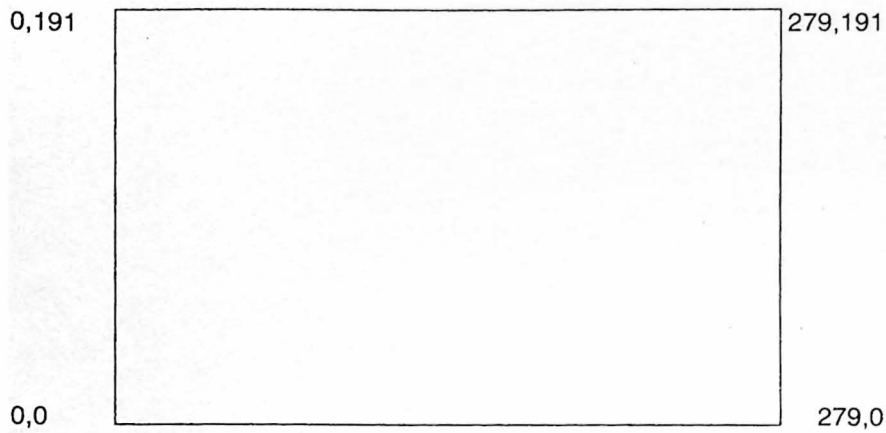
Die Grafik-UNIT wird aufgerufen durch die Anweisung

```
USES TURTLEGRAPHICS;
```

die gleich nach dem Programmnamen kommen muß.

Bildschirm

Der Bildschirm des APPLE ist dann im Grafik-Modus aufgeteilt in 280 x 192 Punkte. Wir können auf einzelne Punkte durch ein Koordinatensystem zugreifen, das wir uns über den Bildschirm gelegt denken:



Bildschirm-aufteilung

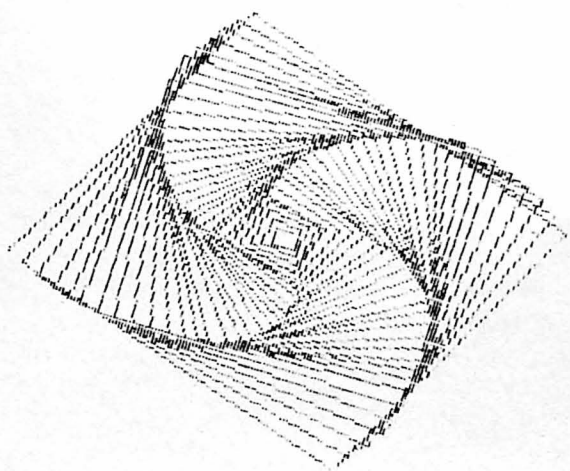
Auf dieser Zeichenfläche ist sowohl direkte Bewegung zu einem Punkt als auch relative Bewegung des Turtle von einem aktuellen Standort aus möglich. Der Turtle (Schildkröte, Zeichenstift) hat stets eine Farbe und eine Richtung. Beide erkennt man allerdings nur, wenn der Turtle bewegt wird.

Text- und Grafik-„Seiten“

Zum weiteren Verständnis der Grafikbefehle und ihrer Funktion müssen wir noch wissen, daß es eine sogenannte Grafik- und eine Textseite gibt. Das, was wir vom Text und von der Grafik auf dem Bildschirm sehen, ist nichts anderes als ein speziell reservierter Speicher. Text- und Grafik-Bildschirmspeicher sind jedoch voneinander getrennt. So haben wir die Möglichkeit, zwischen beiden Seiten mit geeigneten Befehlen hin- und herzuschalten, ohne den jeweiligen Inhalt der Seite zu löschen.

8.2 Befehle der UNIT TURTLEGRAPHICS

Whileplot



Programm

Diese Grafik wurde durch das folgende Programm erzeugt, dessen Anweisungen wir im Laufe des Paragraphen besser verstehen werden:

```
PROGRAM Whileplot;
  USES TURTLEGRAPHICS;

  CONST Anfangsseite = 5;
        Aenderung = 1;
        Winkel = 89;
        Maxseite = 120;

  VAR Seite : INTEGER;

  BEGIN
    INITTURTLE;
    VIEWPORT (0, 279, 0, 191);
    PENCOLOR (WHITE);
    Seite := Anfangsseite;
    WHILE Seite < Maxseite DO BEGIN
      MOVE (Seite);
      TURN (Winkel);
      Seite := Seite + Aenderung
    END; (* von WHILE *)
    READLN (* als Programmstop *)
  END.
```

Aufruf Die UNIT TURTLEGRAPHICS wird im Programm gleich nach dem Programmnamen durch

```
USES TURTLEGRAPHICS;
```

aufgerufen.

Im folgenden werden die Befehle (mit wenigen Ausnahmen recht spezieller Befehle) der Turtle-Grafik aufgeführt und im nächsten Kapitel in kleinen Programmbeispielen verwendet.

Grafik initialisieren INITTURTLE;

Die Anweisung hat keine Parameter. Sie sorgt dafür, daß die Ausgabe auf den Grafikbildschirm umgeschaltet wird, der Bildschirm gelöscht wird und der Turtle (Schildkröte, Zeichenstift) in die Mitte des Bildschirms gesetzt wird und nach rechts zeigt. Die Farbe des Turtle wird auf NONE (keine Farbe) gesetzt.

Grafikseite GRAFMODE;

Die Anweisung hat keine Parameter. Sie bewirkt, daß die Ausgabe auf den Grafikbildschirm umgeschaltet wird, ohne den Bildschirmspeicher, in dem die Grafik sich befindet, zu löschen (im Gegensatz zu INITTURTLE). D.h., eine schon vorhandene Grafik wird wieder sichtbar gemacht.

Textseite TEXTMODE;

Die Anweisung hat keine Parameter. Sie bewirkt, daß die Ausgabe auf den Textbildschirm umgeschaltet wird. Schon vorhandene Texte werden wieder sichtbar.

VIEWPORT (Links, Rechts, Unten, Oben);

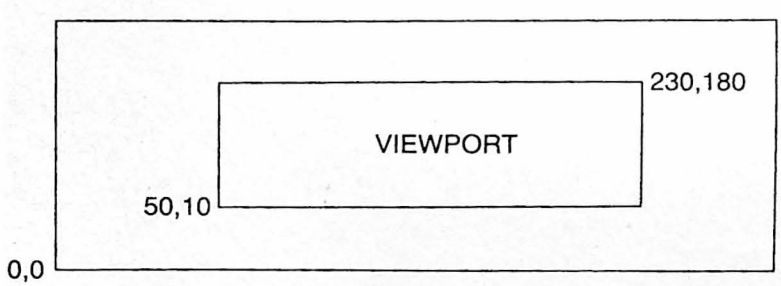
Bildschirmfenster

Die Anweisung hat vier Parameter, die vom Datentyp INTEGER sind. VIEWPORT setzt ein Bildschirmfenster, dessen linker, rechter, unterer und oberer Rand durch die entsprechenden X- und Y-Koordinaten angegeben wird.

```
VIEWPORT ( 50, 230, 10, 180 );
```

bewirkt, daß Punkte nur mit X-Koordinaten zwischen 50 und 230 und Y-Koordinaten zwischen 10 und 180 gezeichnet werden.

VIEWPORT



Hinweis: Punkte außerhalb des VIEWPORT werden zwar nicht gezeichnet, sind jedoch korrekt. Wird aber kein VIEWPORT gesetzt und über den Bildschirmrand gezeichnet, so kann es zu recht merkwürdigen Effekten kommen. Daher möglichst immer einen VIEWPORT (z. B. VIEWPORT (0,279,0,191) für den ganzen Bildschirm) setzen.

PENCOLOR (Farbe);

Farben

Die Anweisung hat einen Parameter von einem vordefinierten Datentyp namens SCREENCOLOR. In der UNIT TURTLEGRAPHICS ist nämlich folgende Typdefinition enthalten:

```
SCREENCOLOR = (NONE, WHITE, BLACK, REVERSE, RADAR,
               BLACK1, GREEN, VIOLET, WHITE1,
               BLACK2, ORANGE, BLUE, WHITE2);
```

Dies ist ein Aufzählungstyp (Band 2), d. h., alle möglichen Werte werden aufgezählt.

PENCOLOR gibt dem Turtle (dem Zeichenstift) eine Farbe, in der er bei jeder Bewegung einen Strich hinterläßt.

Die Farben sind:

WHITE	–	Weiß
WHITE1	–	Weiß1 (2 Punkte breit mit Grün und Violett)
WHITE2	–	Weiß2 (2 Punkte breit mit Orange und Blau)
BLACK	–	Schwarz
BLACK1	–	Schwarz1 (2 Punkte breit mit Grün und Violett)
BLACK2	–	Schwarz2 (2 Punkte breit mit Orange und Blau)

GREEN – Grün
VIOLET – Violett
ORANGE – Orange
BLUE – Blau

Bei Schwarzweiß-Monitoren sollte nur WHITE und BLACK verwendet werden.

Drei weitere Farben sind:

NONE – keine Farbe (also auch nicht Schwarz!)
REVERSE – Umkehrfarbe zum jeweiligen Untergrund. Beispiel: Weiß auf schwarzem und Schwarz auf weißem Untergrund.
RADAR – Farbe für spätere Anwendung (Handbuch).

Bildschirm füllen

FILLSCREEN (Farbe);

Die Anweisung hat einen Parameter vom Datentyp SCREENCOLOR (siehe PENCOLOR). Der gesamte VIEWPORT (das Bildschirmfenster) wird mit der angegebenen Farbe gefüllt.

Winkel einstellen (absolut)

TURNTO (n);

Die Anweisung hat einen Parameter vom Datentyp INTEGER. Der Turtle zeigt nach Ausführung dieser Anweisung in die angegebene Richtung (in Grad), wobei die Richtung der X-Achse (nach rechts) null Grad entspricht.

Bemerkung: Positive Winkel bewirken eine Drehung gegen den Uhrzeigersinn, negative umgekehrt. Winkel können Werte zwischen −360 und +360 annehmen. Werte außerhalb dieses Bereichs werden modulo 360 behandelt.

Weiterdrehen (relativ)

TURN (n);

Die Anweisung hat einen Parameter vom Datentyp INTEGER. Der Turtle wird, ausgehend von seiner aktuellen Richtung, um einen Winkel von n Grad weitergedreht.

Weiterbewegen (relativ)

MOVE (n);

Die Anweisung hat einen Parameter vom Datentyp INTEGER. Der Turtle bewegt sich in seiner aktuellen Richtung um n Einheiten weiter.

Bewegen (absolut)

MOVETO (X, Y);

Die Anweisung hat zwei Parameter vom Datentyp INTEGER. Der Turtle wird von seiner aktuellen Position geradlinig zu einem Punkt mit den Koordinaten (X,Y) bewegt und zeigt nachher weiter in dieselbe Richtung wie vorher.

X-Koordinate

TURTLEX;

Die Funktion TURTLEX erhält einen Wert vom Datentyp INTEGER und kann einer Variablen zugewiesen werden. Sie ergibt die aktuelle X-Position des Turtle.

Dreieck

TURTLEY;

Y-Koordinate

Die Funktion TURTLEY erhält einen Wert vom Datentyp INTEGER und kann einer Variablen zugewiesen werden. Sie ergibt die aktuelle Y-Position des Turtle.

TURTLEANG; ***Winkel***

Die Funktion TURTLEANG erhält einen Wert vom Datentyp INTEGER und kann einer Variablen zugewiesen werden. Sie ergibt den aktuellen Winkel der Richtung des Turtle.

SCREENBIT (X, Y); ***Punkttest***

Die Funktion SCREENBIT hat zwei Parameter vom Datentyp INTEGER und ergibt einen Wert vom Datentyp BOOLEAN. Sie erhält den Wert TRUE, wenn ein Punkt mit den Koordinaten (X,Y) eine Farbe enthält, d.h. nicht BLACK ist; andernfalls FALSE.

Übungen:

8.2.1 Was ist der Unterschied zwischen den Befehlen TURN und TURNTO bzw. MOVE und MOVETO?

8.2.2 Ändern Sie das Programm „Whileplot" so, daß der Benutzer die Form der Spirale selber wählen kann.

8.3 Kleine Programme mit Turtles

1. Zeichnen eines Dreiecks

Wir wollen ein Programm schreiben, das ein gleichseitiges Dreieck mit wählbarer Seitenlänge an eine bestimmte Stelle des Bildschirms zeichnet. ***Aufgabe (Dreieck)***

Dazu brauchen wir die Eingabe der entsprechenden Größen und müssen sodann den Turtle ohne Farbe an die Anfangsposition bewegen. Da der Turtle gleich nach rechts zeigt, können wir sofort mit dem Zeichnen beginnen. Dreimal muß der Turtle um die Seitenlänge vorwärts bewegt und dabei um 120 Grad gedreht werden.

Frage: Warum 120 Grad und nicht 60 Grad? (Skizze!)

Nach dem Programmlauf würde der Rechner wieder auf den Textbildschirm schalten, um die Kommandozeile zu zeigen. Wir hätten gar keine Zeit, unser Dreieck zu bewundern. Daher fügen wir zum Schluß die Anweisung READLN an, die auf irgendeine Eingabe wartet. Das Programm läßt sich dann durch Drücken der Return-Taste beenden.

Blockdiagramm

Programm DREIECK
Benutzt Turtlegrafik
Variablen Seite, X, Y, I : ganze Zahl
Anfang
Eingabe der Werte für Seite und X,Y
Grafik vorbereiten und Bildschirmfenster setzen
Farbe auf NONE setzen
Bewege Turtle nach X,Y
Farbe auf WHITE setzen
Zähle I von 1 bis 3 und tue
Bewege Turtle um Seite weiter
Drehe Turtle um 120 Grad
Warte auf Tastendruck
Schalte auf Textbildschirm
Ende.

Das Programm:

Programm

```
PROGRAM Dreieck;

  USES TURTLEGRAPHICS;

  VAR Seite, X, Y, I : INTEGER;

  BEGIN
    WRITELN ('Dreieck zeichnen: ');
    WRITELN ('Ende mit <Return> ');
    WRITE ('Eingabe Seitenlaenge: ');
    READLN (Seite);
    WRITE ('X-Position: ');
    READLN (X);
    WRITE ('Y-Position: ');
    READLN (Y);

    INITTURTLE;  (* Grafik vorbereiten *)
    VIEWPORT (0,279,0,191);  (* Fenster auf ganzen Bildschirm setzen *)
    PENCOLOR (NONE);
    MOVETO (X,Y);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN  (* Dreieck zeichnen *)
      MOVE (Seite);
      TURN (120)
    END; (* FOR *)
    READLN; (* zum Anhalten *)
    TEXTMODE
  END.
```

2. Zeichnen eines Vielecks

Aufgabe (Vieleck)

Wir wollen das Programm zum Zeichnen eines Dreiecks etwas erweitern, um beliebige regelmäßige Vielecke mit vorgegebener Eckenzahl zu zeichnen.

Fensterbild

Der einzige Unterschied ist der, daß wir nun nicht 3mal, sondern n-mal die Seite zeichnen und nicht um 120 Grad, sondern um 360 DIV n Grad drehen.

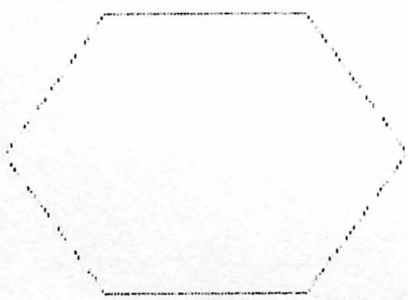
Die Änderungen sind so geringfügig, daß hier auf eine Planung verzichtet werden kann.

Das Programm:

Programm

```
PROGRAM Vieleck;

  USES TURTLEGRAPHICS;

  VAR Seite, X, Y, I, N : INTEGER;

  BEGIN
    WRITELN ('Vieleck zeichnen: ');
    WRITELN ('Ende mit <Return> ');
    WRITE ('Eingabe der Eckenzahl: ');
    READLN (N);
    WRITE ('Eingabe Seitenlaenge: ');
    READLN (Seite);
    WRITE ('X-Position: ');
    READLN (X);
    WRITE ('Y-Position: ');
    READLN (Y);

    INITTURTLE;  (* Grafik vorbereiten *)
    VIEWPORT (0,279,0,191);  (* Fenster auf ganzen Bildschirm setzen *)
    PENCOLOR (NONE);
    MOVETO (X,Y);
    PENCOLOR (WHITE);
    FOR I:=1 TO N DO BEGIN  (* Vieleck zeichnen *)
      MOVE (Seite);
      TURN (360 DIV N)
    END; (* FOR *)
    READLN; (* zum Anhalten *)
    TEXTMODE
  END.
```

Ergebnis



3. Fensterbild mit VIEWPORT

Aufgabe (Fensterbild)

Im folgenden Programm sollen durch wiederholtes Füllen und Verkleinern des Bildschirmfensters ineinandergeschachtelte Rechtecke gezeichnet werden. Zur grafischen Untermalung (und zum Verständnis der Farbe REVERSE) wird vorher ein Dreieck gezeichnet, dessen Konturen später noch sichtbar sind.

Fensterbild

Blockdiagramm

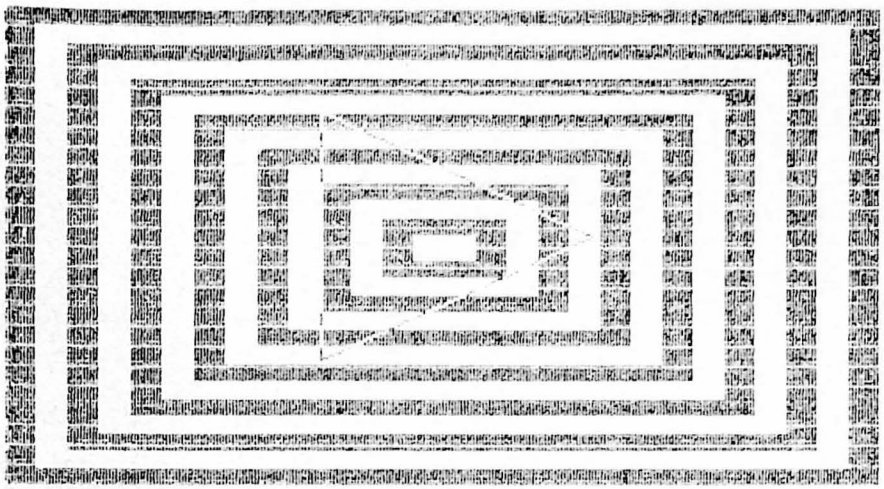
Programm FENSTERBILD	
Benutzt Turtlegrafik	
Variablen XU, XO, YU, YO, I : ganze Zahl	
Anfang	
Grafik vorbereiten	
Farbe auf NONE setzen	
Turtle zu (100,50) bewegen und nach 30 Grad drehen	
Farbe auf WHITE setzen	
Zähle I von 1 bis 3 und tue	
	Bewege Turtle um 100 Punkte
	Drehe Turtle um 120 Grad
XU:=0 XO:=279 YU:=0 YO:=191	
Solange (XO−XU>0) oder (YO−YU>0) tue	
	Setze Bildschirmfenster auf (XU,XO,YU,YO)
	Fülle Bildschirmfenster mit REVERSE
	Enge Bildschirmfenster in der Breite um 20 und in der Höhe um 14 Punkte ein
Warte auf Tastendruck	
Schalte auf Text	
Ende.	

Das Programm:

Programm

```
PROGRAM Fensterbild;

  USES TURTLEGRAPHICS;

  VAR XU, XO, YU, YO, I : INTEGER;

  BEGIN
    INITTURTLE;
    PENCOLOR (NONE);
    MOVETO (100,50);
    TURNTO (30);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN
      MOVE (100);
      TURN (120)
    END; (* FOR *)
    XU:=0; XO:=279; YU:=0; YO:=191;
    WHILE (XO-XU>0) OR (YO-YU>0) DO BEGIN
      VIEWPORT (XU,XO,YU,YO);
      FILLSCREEN (REVERSE);
      XU:=XU+10;
      XO:=XO-10;
      YU:=YU+7;
      YO:=YO-7
    END; (* WHILE *)
    READLN;
    TEXTMODE
  END.
```

Fensterbild



Ergebnis

Übungen:

8.3.1 Schreiben Sie ein Programm, das es erlaubt, durch wiederholte Eingabe von X- und Y-Werten im Dialog mit dem Computer auf dem Bildschirm zu zeichnen.

8.3.2 Erweiterung von 8.3.1: Änderung der Farbe soll ermöglicht werden.

8.3.3 Ein Programm soll das Zeichnen mit Hilfe der Game-Paddles erlauben. Durch Bedienung der Druckknöpfe soll gezeichnet bzw. nicht gezeichnet werden.

Die aktuelle Position des Punkts auf dem Bildschirm soll durch ein Fadenkreuz mitwandern.

Tip: Benutzen Sie auch die UNIT APPLESTUFF, siehe Kap. 8.4.

8.3.4 Ein Programm soll zufällige Linien (zufällig in Zeichenrichtung und Länge) auf den Bildschirm zeichnen. Beendet wird das Zeichnen beim Erreichen des Bildschirmrands.

Tip: Benutzen Sie auch die UNIT APPLESTUFF, siehe Kap. 8.4.

8.4 Andere UNITs

Hinweis: Wenn mehrere UNITs in einem Programm verwendet werden sollen, so werden sie hinter USES durch Kommata getrennt aufgeführt. Beispiel: .

mehrere UNITs

```
USES APPLESTUFF, TURTLEGRAPHICS, TRANSCEND;
```

Folgende Möglichkeiten für ein kreatives Programmieren sind in anderen UNITs der SYSTEM.LIBRARY enthalten:

APPLESTUFF

Mit

```
USES APPLESTUFF;
```

erhält man die zusätzlichen Anweisungen:

Zufallszahl RANDOM;

Die Funktion RANDOM hat keinen Parameter und ergibt eine Zufallszahl zwischen 0 und 32767 (MAXINT).

Tip: Wird z. B. eine Zufallszahl zwischen 0 und 9 gesucht, so berechnet man RANDOM MOD 10.

Wird eine Zufallszahl zwischen i und j gewünscht, so benutzen wir die Formel: i + RANDOM MOD (j−i+1).

neue Zufallszahl RANDOMIZE;

Die Anweisung hat keine Parameter. Sie sorgt dafür, daß eine neue Serie von Zufallszahlen produziert wird. Ohne RANDOMIZE würde bei jedem Programmlauf dieselbe Serie von Zufallszahlen benutzt, was zu Testzwecken gut gebraucht werden kann.

Tastendruck KEYPRESS;

Die Funktion KEYPRESS hat keine Parameter und ergibt einen Wert vom Datentyp BOOLEAN. Sie erhält den Wert TRUE, wenn eine Taste auf der Tastatur seit dem Programmstart oder seit dem letzten READ (READLN) gedrückt wurde.

Achtung: KEYPRESS liest selber kein Zeichen von der Tastatur!

Handregler PADDLE (n);

Die Funktion hat einen Parameter vom Datentyp INTEGER. n kann die Werte 0, 1, 2 und 3 annehmen. Größere Werte werden modulo 4 behandelt. Das Ergebnis ist vom Datentyp INTEGER.

Die Funktion PADDLE wählt einen der vier möglichen Handregler aus (bestehend aus je einem 150K-Potentiometer) und erhält einen Wert zwischen 0 und 255, abhängig von der Einstellung des Handreglers.

Druckknopf BUTTON (n);

Die Funktion hat einen Parameter vom Datentyp INTEGER. n kann die Werte 0, 1 und 2 annehmen. Größere Werte werden modulo 3 behandelt. Das Ergebnis ist vom Datentyp BOOLEAN.

Die Funktion BUTTON wählt einen der drei möglichen Druckknöpfe an den Handreglern aus und erhält den Wert TRUE, wenn der Druckknopf gedrückt wird; sonst FALSE.

Hinweis: Mit PADDLE und BUTTON haben wir die Möglichkeit, über den Anschluß der Handregler vier Analogeingänge (0 bis 150K)

und drei Digitaleingänge (0V oder +5V) zu verwenden. Das ist interessant für einfache Projekte zur Meßwerterfassung.

TTLOUT (n, b);

TTL-Ausgang

Die Anweisung hat zwei Parameter. n ist vom Datentyp INTEGER und kann die Werte 0, 1, 2 und 3 annehmen. Größere Werte werden modulo 4 behandelt, b ist vom Datentyp BOOLEAN.

Mit TTLOUT wird einer der vier TTL-Ausgänge angewählt und ihm ein Wert TRUE oder FALSE übergeben.

Mit TRUE liegt an dem entsprechenden Ausgang +5V, mit FALSE 0V an. Ebenfalls interessant für rechnergesteuerte Versuchsauswertungen.

NOTE (h, l);

Tonerzeugung

Die Anweisung hat zwei Parameter vom Datentyp INTEGER. h kann Werte zwschen 0 und 50, 1 Werte zwischen 0 und 255 annehmen. NOTE erzeugt einen Ton mit der Tonhöhe h und der Länge l. h=0 ergibt eine Pause.

h zwischen 2 und 48 entsprechen ungefähr Tonhöhen nach der Tonleiter. NOTE (1,1); ergibt ein klickendes Geräusch.

Eine kleine Tonleiter:

Musik-Programm

```
PROGRAM Tonleiter;

  USES APPLESTUFF;

  VAR h, l : INTEGER;

  BEGIN
    l:=100;
    FOR h:=12 TO 24 DO NOTE (h,l)
  END.
```

TRANSCEND

Mit

Winkelfunktionen und andere

```
USES TRANSCEND;
```

erhält man die zusätzlichen Anweisungen:

```
SIN (x)
COS (x)
EXP (x)
ATAN (x)
LN (x)
LOG (x)
SQRT (x)
```

Die obengenannten Funktionen sind in Kap. 3.5 beschrieben.

8.5 Zusammenfassung

Programmieren soll Spaß machen!

Mit den im letzten Kapitel gelernten Grafikroutinen können wir viel Spaß haben und viele Programme durch entsprechende Grafikausgaben wertvoll bereichern.

Oft sagt eine übersichtliche Tabelle mehr über Zusammenhänge aus als drei Seiten Text. Auch die Bereiche der mathematischen Darstellungen und der Spiele stehen uns jetzt offen.

Mit weiteren UNITs haben wir auch die Möglichkeit, mit der Außenwelt in Kontakt zu treten. So ließe sich z. B. mit dem Game-I-O-Port des APPLE (und natürlich auch mit jedem anderen Port eines Computers) eine elektrische Eisenbahn steuern. Das macht sicher einem Eisenbahnhobbyisten Spaß, kann aber auch ein Projekt im Informatikunterricht sein.

Besser geht's mit Prozeduren

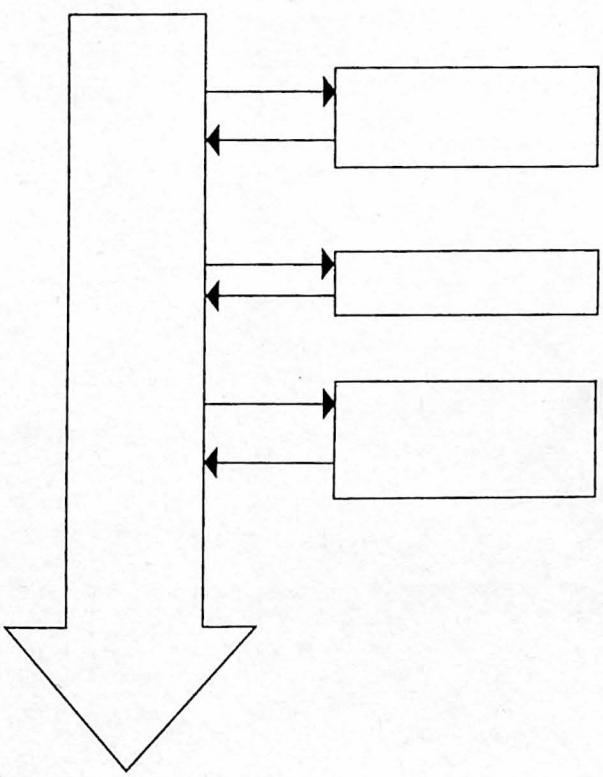
Kapitel 9

In diesem Kapitel beschäftigen wir uns ausführlich mit Unterprogrammen, sogenannten Prozeduren. In vorhergehenden Kapiteln haben wir einfache Prozeduren schon vereinzelt verwendet, um uns die Arbeit zu erleichtern oder das Programm übersichtlicher zu gestalten.

Allerdings handelte es sich nur um einfache Prozeduren ohne Variablenübergabe. Nun werden wir lernen, wie man elegant und vielfältig Prozeduren einsetzt, um zu einem übersichtlichen und systematischen Programmierstil zu kommen.

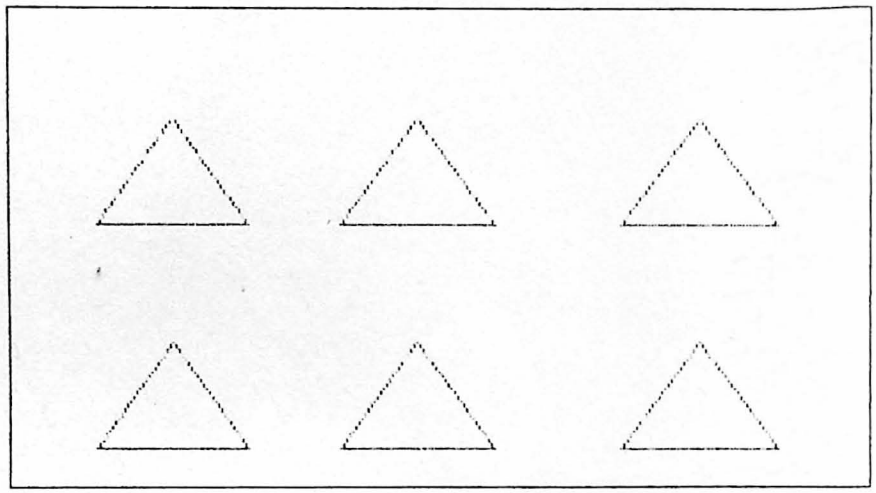
Nach dem Lesen dieses Kapitels

- kennen wir die Struktur und Syntax von Prozeduren.
- können wir lokale und globale Variablen unterscheiden.
- kennen wir die Rangfolge gleichnamiger Variablen.
- können wir Prozeduren mit Variablenübergaben benutzen.
- wissen wir den Unterschied zwischen der Variablenübergabe call-by-value und call-by-variable.
- wissen wir, was man unter systematischem Programmieren versteht.



9.1 Wenn wir das Dreieck immer wieder zeichnen wollen. Prozeduren und Prozedurkonzept

6 Dreiecke



Programm dazu (ganz schön umständlich)

```
PROGRAM Dreieck;

  USES TURTLEGRAPHICS;

  CONST Seite = 50;
  VAR I : INTEGER;

  BEGIN
    INITTURTLE;
    PENCOLOR (NONE);
    MOVETO (0,0);
    PENCOLOR (WHITE);
    MOVETO (279,0);
    MOVETO (279,191);
    MOVETO (0,191);
    MOVETO (0,0);
    PENCOLOR (NONE);
    VIEWPORT (1,278,1,190);

    MOVETO (10,10);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN
      MOVE (Seite);
      TURN (120)
    END;
    PENCOLOR (NONE);
    MOVETO (90,10);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN
      MOVE (Seite);
      TURN (120)
    END;
    PENCOLOR (NONE);

    MOVETO (180,10);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN
      MOVE (Seite);
      TURN (120)
    END;
    PENCOLOR (NONE);

    MOVETO (10,100);
    PENCOLOR (WHITE);
```

(Fortsetzung des Programms)

```
    FOR I:=1 TO 3 DO BEGIN
      MOVE (Seite);
      TURN (120)
    END;
    PENCOLOR (NONE);

    MOVETO (90,100);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN
      MOVE (Seite);
      TURN (120)
    END;
    PENCOLOR (NONE);

    MOVETO (180,100);
    PENCOLOR (WHITE);
    FOR I:=1 TO 3 DO BEGIN
      MOVE (Seite);
      TURN (120)
    END;
    PENCOLOR (NONE);

    READLN
  END.
```

Also wirklich ...

Es geht eleganter ...

Man muß schon ziemlich viel Zeit haben, um dieses Programm einzutippen. Sechsmal haben wir nun fast den gleichen Satz von Programmzeilen geschrieben, in dem die einzelnen Dreiecke gezeichnet werden. Nur die Zeile MOVETO (,); ist von Dreieck zu Dreieck verschieden, da die Dreiecke an die Positionen (10,10), (90,10), (180,10), (10,100), (90,100) und 180,100) gezeichnet werden sollen.

Außerdem wird am Anfang des Programms ein Rahmen um das Bild gezeichnet, den wir in anderen Programmen ebenfalls verwenden können.

Eine intelligentere und einfachere Version des Programms Dreieck ist die folgende:

Blockdiagramm

Programm DREIECK
Benutzt Turtlegrafik
Konstante Seite = 50
Variable I : ganze Zahl
Anfang
RAHMEN
Bewege nach (10,10)
FIGUR
Bewege nach (90,10)
FIGUR
Bewege nach (180,10)
FIGUR
Bewege nach (90,100)

(Fortsetzung des Blockdiagramms)

FIGUR
Bewege nach (180,100)
FIGUR
Warte auf Returntaste
Ende.

Prozeduren

Dabei sind die Anweisungen RAHMEN und FIGUR sicher keine Pascal-Befehle. Vielmehr handelt es sich um die Prozeduren mit Namen RAHMEN und FIGUR.

Die Prozedur FIGUR wird vom Hauptprogramm sechsmal aufgerufen, nachdem der Turtle an die entsprechende Bildschirmposition gebracht wurde. Bei unserer Planung tun wir so, als gäbe es die Prozeduren schon. In Wirklichkeit müssen wir sie erst schaffen:

Planung einer Prozedur

Prozedur FIGUR	
Anfang	
Farbe Weiß	
Zähle I von 1 bis 3 und tue	
	Bewege um Seite
	Drehe um 120 Grad
Keine Farbe	
Ende;	

Aufgabe: Planen Sie die Prozedur Rahmen.

Das Programm mit Prozeduren:

Programm mit Prozeduren

```
PROGRAM Dreieck;

  USES TURTLEGRAPHICS;

  CONST Seite = 50;
  VAR I : INTEGER;

  PROCEDURE Rahmen;
    BEGIN
      INITTURTLE;
      PENCOLOR (NONE);
      MOVETO (0,0);
      PENCOLOR (WHITE);
      MOVETO (279,0);
      MOVETO (279,191);
      MOVETO (0,191);
      MOVETO (0,0);
      PENCOLOR (NONE);
      VIEWPORT (1,278,1,190)
    END; (* von Rahmen *)

  PROCEDURE Figur;
    BEGIN
      PENCOLOR (WHITE);
      FOR I:=1 TO 3 DO BEGIN
```

(Fortsetzung des Programms)

```
        MOVE (Seite);
        TURN (120)
      END;
      PENCOLOR (NONE)
    END; (* von Figur *)

  BEGIN (* Hauptprogramm *)
    Rahmen;
    MOVETO (10,10);
    Figur;
    MOVETO (90,10);
    Figur;
    MOVETO (180,10);
    Figur;
    MOVETO (10,100);
    Figur;
    MOVETO (90,100);
    Figur;
    MOVETO (180,100);
    Figur;
    READLN
  END.
```

Wir haben es hier vorerst mit einer sehr einfachen Art der Prozedur zu tun. Unsere Prozeduren besitzen nur die Variablen und Konstanten, die im Hauptprogramm definiert sind.

globale Variablen

Variablen und Konstanten des Hauptprogramms sind also im Hauptprogramm *und* in den Prozeduren gültig.

Wir nennen sie *globale* Variablen.

In diesem Paragraphen wollen wir es erst einmal bei dieser einfachen Prozedurart belassen. Hier soll Grundsätzliches über Prozeduren geklärt werden, bevor wir andere Prozedurarten erarbeiten.

mehrere Prozeduren

Wir können in Pascal mehrere Prozeduren verwenden und durch Aufruf des Prozedurnamens abarbeiten lassen.

Reihenfolge des Aufrufs

Es kann auch von einer Prozedur aus eine andere Prozedur aufgerufen werden. Dazu gibt es aber eine sehr wichtige Vorschrift: Die aufgerufene Prozedur muß im Programmtext *vor* der aufrufenden Prozedur geschrieben werden.

Beispiel

Beispiel:

```
PROGRAM Na_so_was;

  PROCEDURE Falsch; (* Die ist wirklich falsch *)
    BEGIN
      Hallo          (* Hallo existiert hier noch nicht *)
    END;

  PROCEDURE Hallo;
    BEGIN
      WRITELN ('Hallo!')
    END;

  PROCEDURE Richtig; (* Die ist aber richtig *)
    BEGIN
      Hallo          (* Hallo existiert schon *)
    END;
```

(Fortsetzung des Programms)

```
BEGIN (* Hauptprogramm *)
  Falsch;
  Hallo;
  Richtig;
  Hallo
END.
```

Prozeduren können geschachtelt werden:

Schachtelung von Prozeduren

```
PROGRAM Schachtel;

  PROCEDURE Text;

    PROCEDURE Linie;
      VAR I : INTEGER;
      BEGIN
        FOR I:=1 TO 40 DO WRITE ('-');
        WRITELN
      END; (* von Linie *)

    BEGIN (* von Text *)
      Linie;
      WRITELN ('Dies ist ein Test ');
      Linie
    END; (* von Text  *)

  BEGIN  (* Hauptprogramm *)
    Text
  END.
```

Bemerkungen

Jedes Programm kann durch leichte Veränderungen zu einer Prozedur gemacht und in anderen Programmen verwendet werden.

Häufig gebrauchte Prozeduren sollte man sich als gesonderte Text-Files abspeichern und mit dem EDITOR-Befehl C für „Copy" und F für „from file" in das zu schreibende Programm hineinkopieren.

Eine Prozedur (oder das Hauptprogramm, das vom System auch als Prozedur angesehen wird!) kann schnell zu groß werden. Dies passiert vor allem, wenn viel Text verwendet wird. In diesem Fall müssen mehrere kleine Prozeduren daraus gemacht werden.

Definition Prozedur

Allgemeines zu Prozeduren:

- Eine Prozedur ist ein Unterprogramm (Teilprogramm), das durch Nennung des Prozedurnamens aufgerufen wird.
- Variablen des Hauptprogramms (globale Variablen) sind auch in Prozeduren gültig.
- Eine Prozedur kann eine andere aufrufen. Dabei gilt: Die aufgerufene Prozedur muß vor der aufrufenden im Programmtext stehen.
- Prozeduren können geschachtelt werden.

Verlassen einer Prozedur

Hinweis für UCSD-Pascal: Prozeduren können mit der Anweisung

EXIT (prozedurname);

global/lokal

verlassen werden. Mit dieser Anweisung wird die Prozedur vorzeitig beendet, und das Programm springt zu der Stelle zurück, von der aus die Prozedur aufgerufen wurde.

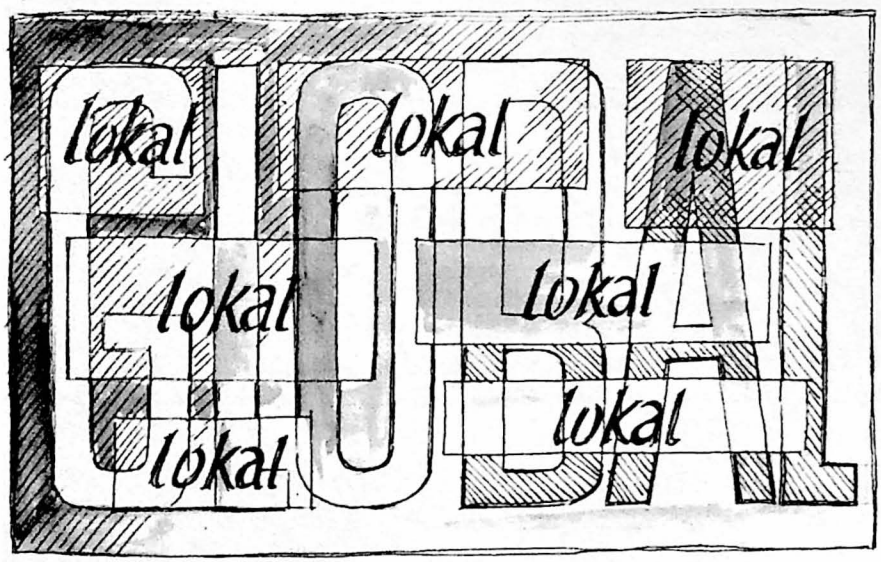
Wir wollen diese Anweisung aber im Sinne einer systematischen Programmierung möglichst vermeiden (so, wie Sprünge mit GOTO in diesem Buch erst gar nicht auftreten).

Übungen:

9.1.1 Schreiben Sie eine Prozedur „Lies", die ein Wort von der Tastatur ohne Drücken der <Return>-Taste einliest und als Wort dreimal hintereinander ausgibt.

9.1.2 Schreiben Sie das Programm „Bruchrechnen" aus Kap. 7.2 unter Benutzung von Prozeduren um.

9.2 Prozeduren können ihre eigenen Variablen haben



Deklarationsteil der Prozedur

Wir können Prozeduren einen eigenen Deklarationsteil mitgeben, in dem nur für die entsprechende Prozedur gültige Konstanten und Variablen erklärt werden. Der Prozedurkopf ist also ähnlich zu gestalten wie der Programmkopf:

```
PROCEDURE name;
   CONST ...;
   VAR   ...;

   BEGIN
      ...
   END;
```

Prozedur-Variablen

kein USES

Hinweis: Im Prozedurkopf darf kein Aufruf einer UNIT aus einer LIBRARY (siehe Kap. 8) mit USES erfolgen. Dies ist nur im Programmkopf für das gesamte Programm erlaubt.

lokal/global

→ Variablen und Konstanten, die in einer Prozedur deklariert werden, nennt man *lokal* (im Gegensatz zu den globalen Variablen und Konstanten des Hauptprogramms).

Gültigkeit

→ Lokale Variablen und Konstanten können *nur* in der Prozedur verwendet werden, in der sie definiert sind. In anderen Prozeduren und im Hauptprogramm sind sie folglich nicht zu verwenden.

Schachtelung

→ Bei geschachtelten Prozeduren gilt: Die Variablen der äußeren Prozedur sind global zu denen der inneren Prozedur, die Variablen der inneren Prozedur sind lokal.

Konflikte (gleiche Namen)

→ Konflikte: Lokale Variablen können sogar die gleichen Namen haben wie globale Variablen. In diesem Fall setzt sich die lokale Vereinbarung durch.

Schleifen

→ Schleifenvariablen in FOR-Schleifen sollten (in vielen Pascal-Versionen: müssen) lokale Variablen sein.

Aufgabe (Datum)

Wir wollen nun ein kleines Beispielprogramm planen und ausführen. Aufgabe des Programms: Nach der Eingabe eines Datums ermittelt das Programm, um welchen Wochentag es sich bei diesem Datum handelt. Außerdem: Ein Freitag, der 13., wird besonders erwähnt.

Lösungshilfen

Zur Lösung des Problems existiert eine Formel, vorausgesetzt, der Zeitraum der einzugebenden Daten wird eingeschränkt auf Daten zwischen 1701 und 2099. Dann läßt sich vom 1.3.1700 an eine ganze Zahl von 0 aufwärtszählen. Da der 1.3.1700 ein Sonntag war, können wir die Zahl n der Tage seit dem 1.3.1700 durch 7 teilen und den Rest betrachten. Wenn der Rest 0 ist, haben wir Sonntag. Bei einem Rest von 1 Montag usw.

Da die Zahl der Tage seit dem 1.3.1700 natürlich rasch sehr groß wird, läßt sich n nicht mit dem Datentyp INTEGER beschreiben. So müssen wir Long INTEGER wählen. Bei diesem Datentyp gibt es jedoch nicht den MOD-Befehl. Daher wird der Rest „zu Fuß" ausgerechnet.

Die Formel für die Zahl der Tage n lautet:

```
n = TRUNC (365.25 * g) + TRUNC (30.6)* f) + Tag - 621049
```

wobei gilt:

```
g=Jahr, wenn Monat>2, sonst g=Jahr-1
f=Monat+1, wenn Monat>2, sonst f=Monat+13
```

Für Tage zwischen dem 1. 3. 1700 und dem 28. 2. 1800 muß n um 2 erhöht und für Tage zwischen dem 1. 3. 1800 und dem 28. 2. 1900 um 1 erhöht werden.

Das Datum besteht aus den Teilen: Tag, Monat, Jahr.

Wochentag

Blockdiagramm Hauptprogramm

Programm DATUM
Variablen Tag, Monat, Jahr, Tagnummer: ganze Zahl Frage: Zeichen
Anfang
Wiederhole
Bildschirm löschen
Eingabe
Berechne
Ausgabe
Frage nach weiterer Berechnung
bis Frage = 'N' oder Frage = 'n'
Ende.

Aufgabe: Planen Sie die Prozedur Eingabe!
Achten Sie darauf, daß die Eingabe gültig sein muß.
Das Datum muß in dem vorgegebenen Bereich liegen, bestimmte Monate haben 28 (29), 30 oder 31 Tage.

Blockdiagramm Prozedur BERECHNE

Prozedur BERECHNE	
Konstante Dif=621049	
Variablen Hilfe, n : lange ganze Zahl k : ganze Zahl	
Anfang	
k:=0	
wenn Jahr < 1900	
ja	
k:=1	
wenn Jahr = 1900	
ja	
wenn Monat < 3	
ja	
k:=1	
wenn Jahr < 1800	
ja	
k:=2	
wenn Jahr = 1800	
ja	
wenn Monat < 3	
ja	
k:=2	
wenn Monat > 2	
ja	nein

(Fortsetzung des Blockdiagramms)

Monat:=Monat+1	Monat:=Monat+13 Jahr:=Jahr−1
n:= 36525∗Jahr DIV 100 + 306∗Monat DIV 10 + Tag − Dif + k	
Hilfe := n DIV 7	
n := n − Hilfe∗7	
Tagnummer := TRUNC (n)	
Ende;	

Aufgabe: Planen Sie die Prozedur Ausgabe!

Programm Das Programm:

```
PROGRAM Datum;

  VAR Tag,Monat,Jahr,Tagnummer : INTEGER;
      Frage : CHAR;

  PROCEDURE Eingabe;

    VAR Februar : INTEGER;
        Erfolg : BOOLEAN;

    BEGIN
      WRITELN ('Das Programm ermittelt zu einem beliebigen Datum innerhalb ');
      WRITELN ('des Zeitraumes 1701 bis 2099 den dazugehoerigen Wochentag.');
      WRITELN ('Geben Sie bitte das Datum ein:');
      WRITELN;
      Erfolg := TRUE;
      REPEAT
        IF NOT Erfolg THEN WRITELN ('Falsche Eingabe ! Neu eingeben:');
        WRITE ('Geben Sie den Tag ein:');
        READLN (Tag);
        WRITE ('geben Sie den Monat ein:');
        READLN (Monat);
        WRITE ('Geben Sie das Jahr ein:');
        READLN (Jahr);
        IF Jahr<100 THEN Jahr:=1900+Jahr; (* fuer Schreibfaule *)
        IF (Jahr<1701) OR (Jahr>2099) THEN Erfolg := FALSE;
                                          (* nicht im Bereich *)
        IF Monat>12 THEN Erfolg := FALSE; (* Monat zu gross.*)
        IF (Jahr MOD 4 =0) THEN Februar:=29 ELSE Februar:=28; (* Schaltjahr *)
        IF (Jahr MOD 100 =0) THEN Februar:=28; (* kein Schaltjahr *)
        IF (Jahr MOD 400 =0) THEN Februar:=29; (* Schaltjahr *)
        IF Tag>31 THEN Erfolg := FALSE; (* Tag zu gross *)
          CASE Monat OF
            2 : Erfolg:=(Tag<=Februar);
            4 : Erfolg:=(Tag<=30);
            6 : Erfolg:=(Tag<=30);
            9 : Erfolg:=(Tag<=30);
           11 : Erfolg:=(Tag<=30)
          END; (* von Case *)
      UNTIL Erfolg
    END; (* von Eingabe *)

  PROCEDURE Berechne;

    CONST Dif=621049;

    VAR Hilfe, n : INTEGER[9];
        k, Monat1, Jahr1  : INTEGER;

    BEGIN
      k:=0; Jahr1 := Jahr; Monat1 := Monat;
      IF Jahr<1900 THEN k:=1;
      IF Jahr=1900 THEN IF Monat<3 THEN k:=1;
      IF Jahr<1800 THEN k:=2;
      IF Jahr=1800 THEN IF Monat<3 THEN k:=2;
      IF Monat>2 THEN Monat1:=Monat+1
```

(Fortsetzung des Programms)

```
        ELSE BEGIN
          Monat1:=Monat+13;
          Jahr1:=Jahr-1
        END; (* von Else *)
      n := 36525*Jahr1 DIV 100 + 306*Monat1 DIV 10 + Tag - Dif + k;
      Hilfe := n DIV 7;
      n := n - Hilfe*7;
      Tagnummer:=TRUNC(n)
    END; (* von Berechne *)

  PROCEDURE Ausgabe;

    BEGIN
      PAGE(OUTPUT);
      WRITELN;WRITELN;WRITELN;
      IF (Tagnummer=5) AND (Tag=13) THEN BEGIN
          WRITELN ('An Ihrer Stelle wuerde ich mich vorsehen, denn der ');
          WRITELN (' 13.',Monat,'.',Jahr,' ist ein Freitag.')
        END
      ELSE
        BEGIN
          WRITE ('Der ',Tag,'.',Monat,'.',Jahr,' ist ein ');
          CASE Tagnummer OF
            0 : WRITELN ('Sonntag');
            1 : WRITELN ('Montag');
            2 : WRITELN ('Dienstag');
            3 : WRITELN ('Mittwoch');
            4 : WRITELN ('Donnerstag');
            5 : WRITELN ('Freitag');
            6 : WRITELN ('Samstag')
          END (* von Case *)
        END (* von Else *)
    END; (* von Ausgabe *)

  BEGIN  (* Hauptprogramm *)
    REPEAT
      PAGE(OUTPUT);
      Eingabe;
      Berechne;
      Ausgabe;
      WRITELN;
      WRITE ('Wuenschen Sie eine erneute Berechnung (J/N) ? ');
      READ(Frage)
    UNTIL (Frage = 'N') OR (Frage = 'n')
  END.
```

Definition Prozedur mit lokalen Größen

Prozeduren mit Konstanten- und Variablendeklaration.

```
Procedure name;
   CONST  . . . ;
   VAR    . . . ;

   BEGIN
      . . .
   END;
```

Konstanten und Variablen, die in einer Prozedur deklariert sind, heißen lokal und sind nur in der Prozedur gültig. Konstanten und Variablen, die im Hauptprogramm (Programmkopf) deklariert sind, heißen global und sind überall gültig.

Lokale und globale Variablen und Konstanten können gleiche Namen haben. Im Konfliktfall haben lokale Variablen und Konstanten den Vorrang.

Übung:

9.2.1 Schreiben Sie ein Grafikprogramm mit Prozeduren (und darin lokalen Variablen), das fest vorgegebene Funktionen (z. B. SIN, COS, TAN) in einzulesenden Grenzen zeichnet.

Es soll ein Achsenkreuz gezeichnet und die Teilung der Achsen vom Programm optimiert werden.

9.3 Übergabe von Variablen an Prozeduren

Transportprobleme

Wir haben schon des öfteren Prozeduren gehabt, die mit Variablen des Hauptprogramms arbeiten (mit sogenannten globalen Variablen). Solche Prozeduren sind schlecht zu transportieren. Da man wissen muß, wie die globalen Variablen heißen, läßt sich eine solche Prozedur nicht ohne weiteres in einem beliebigen anderen Programm verwenden. Dies ist aber oft erwünscht, damit häufig gebrauchte Prozeduren (Standardprozeduren) nicht immer wieder geschrieben werden müssen.

Auch an dieser Stelle läßt uns Pascal nicht allein und bietet eine Möglichkeit der Variablenübergabe an eine Prozedur. Dazu wird hinter dem Prozedurnamen eine Liste der Variablen aufgeschrieben, die an die Prozedur übergeben werden können und dann auch müssen.

Beispiel

```
PROCEDURE name (a,b : INTEGER; c : CHAR);
   CONST . . .;
   VAR   . . .;
   BEGIN
      . . .
   END;
```

In diesem Beispiel werden der Prozedur drei Werte (values) übergeben, nämlich a, b und c, wobei die ersten beiden vom Typ INTEGER und die dritte vom Typ CHAR sein muß.

Die Prozedur könnte dann in folgenden Versionen aufgerufen werden:

richtiger Aufruf

```
x:=5;
y:=8;
z:='g';

name(x,y,z);
name(7,9,'e');
name(y,5,'u');
```

Wichtig ist nur zu wissen, daß die so definierte Prozedur name drei Werte bestimmten Typs und bestimmter Reihenfolge braucht. Die Namen der Variablen (a, b, c) tauchen beim Prozeduraufruf überhaupt nicht auf.

Werte übergeben

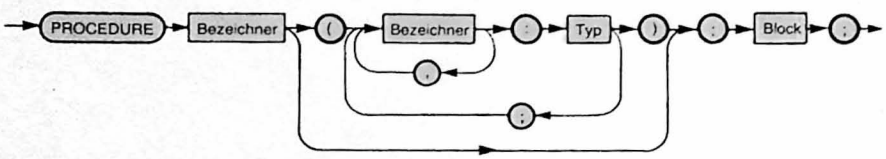
Ein falscher Aufruf wäre allerdings:

falscher Aufruf

```
name('u',y,5);
```

denn der erste und der dritte Wert stimmen nicht mit den in der Prozedur definierten Typen überein.

Allgemein hat der Kopf einer Prozedur mit Variablenübergabe an die Prozedur folgende Form:



Definition Prozedur mit Variablenübergabe

→ Diese Art der Variablenübergabe nennt man call-by-value, da die Prozedur mit einem Wert (value) aufgerufen wird.

call-by-value

→ *Achtung:* Wenn beim Prozeduraufruf eine Variable verwendet wird, so wird ihr Wert durch die Prozedur nicht verändert.

keine Änderung der Variablen

Dazu ein Beispiel:

Beispiel

```
PROGRAM call_by_value;

  VAR a : INTEGER;

  PROCEDURE Test ( x : INTEGER );
    BEGIN
      x := x * 2;
      WRITELN (x)
    END;

  BEGIN (* Hauptprogramm *)
    a := 7;
    WRITELN (a);
    Test (3);
    Test (a);
    WRITELN (a)
  END.
```

Dieses Programm schreibt die Werte 7, 6, 14 und 7 untereinander.

Analyse

Warum?

Zuerst wird der Variablen a der Wert 7 zugeordnet, der ausgegeben wird. Der Aufruf der Prozedur Test mit dem Wert 3 bewirkt, daß das Doppelte von 3 geschrieben wird. Danach wird Test mit dem Wert a (der ja 7 enthält) aufgerufen und das Doppelte von 7 ausgegeben. Da eine Prozedur mit der call-by-value-Variablenübergabe die Variablen des Hauptprogramms nicht ändert, hat a weiterhin den Wert 7. Daher wird diese 7 zuletzt ausgegeben.

Frage: Was ändert sich in der Ausgabe des Programms, wenn wir die Variable x der Prozedur in a umbenennen?
Antwort: Nichts! (siehe Kap. 9.2)

Nun noch ein weiteres Beispiel für ein Programm mit call-by-value-Prozeduren.

Aufgabe Warenpreis

Das folgende Programm soll nach der Eingabe des Nettopreises und des Gewichts einer Ware den Gesamtpreis der Ware berechnen und die Berechnung in einer Tabelle darstellen.

Zum Nettopreis kommen noch 14 % Mehrwertsteuer, ein nach Gewicht gestaffelter Verpackungspreis und, ebenfalls nach Gewicht gestaffelt, Porto hinzu.

Programm

```
PROGRAM Warenpreis;

  VAR Preis, Gewicht : REAL;
      Laenge : INTEGER;
      Frage : CHAR;

  PROCEDURE Linie ( l : INTEGER ; Doppel : BOOLEAN);

    VAR i : INTEGER;
        ch : CHAR;

    BEGIN
      IF Doppel THEN ch := '=' ELSE ch := '-';
      FOR i := 1 TO l DO WRITE (ch);
      WRITELN
    END; (* von Linie *)

  PROCEDURE Tabelle (Netto, Gewicht : REAL);

    CONST MW = 14;
          Pausch = 15;

    VAR Steuer, Brutto, Verpa, Porto, Gesamt : REAL;
        Doppel : BOOLEAN;

    BEGIN
      Steuer := Netto * MW / 100;
      Brutto := Netto + Steuer;
      IF Gewicht<=5 THEN BEGIN Verpa:=1.50;
                               Porto:=4.40
                         END
      ELSE IF Gewicht<=8 THEN BEGIN Verpa:=2.50;
                                    Porto:=6.20
                              END
           ELSE IF Gewicht<=12 THEN BEGIN Verpa:=5.50;
                                          Porto:=8.60
                                    END
                ELSE BEGIN Verpa:=Pausch;
                           Porto:=Pausch
                     END;
      Gesamt := Brutto + Verpa + Porto;
      WRITELN ('Nettopreis der Ware ',Netto:7:2,' DM');
      WRITELN ('dazu ',MW:2,'% MWSt.       ',Steuer:7:2,' DM');
      Linie (30,FALSE);
      WRITELN ('Bruttopreis d. Ware ',Brutto:7:2,' DM');
      Linie (30,TRUE);
      WRITELN ('Gewicht             ',Gewicht:7:2,' kg');
      Linie (30,FALSE);
      WRITELN ('Verpackung          ',Verpa:7:2,' DM');
      WRITELN ('Porto               ',Porto:7:2,' DM');
      Linie (30,FALSE);
      WRITELN ('Gesamtpreis:        ',Gesamt:7:2,' DM');
      Linie (30,TRUE)
    END;  (* von Tabelle *)

  BEGIN (* Hauptprogramm *)
    REPEAT
      PAGE (OUTPUT);
      WRITE ('Eingabe des Nettopreises in DM: ');
      READLN (Preis);
      WRITE ('Eingabe des Gewichts in kg    : ');
```

Procedur call-by-value

(Fortsetzung des Programms)

```
    READLN (Gewicht);
    Tabelle (Preis, Gewicht);
    WRITELN;
    WRITE ('Eine weitere Tabelle (J/N) ? ');
    READ (Frage)
  UNTIL (Frage='N') OR (Frage='n')
END.
```

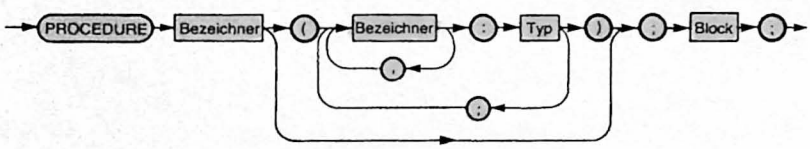
Bei einer Eingabe von 15.5 als Preis und 10 als Gewicht ergibt das Programm folgende Ausgabe:

Ergebnis

Nettopreis der Ware	15.50 DM
dazu 14 % MWSt.	2.17 DM
Bruttopreis der Ware	17.67 DM
Gewicht	10.00 kg
Verpackung	5.50 DM
Porto	8.60 DM
Gesamtpreis	31.77 DM

Definition Prozedur vom Typ call-by-value

Prozedur mit Variablenübergabe vom Typ call-by-value:



Der Prozeduraufruf erfolgt durch Nennung des Namens der Prozedur, gefolgt von den zu übergebenden Werten in einer Klammer. Die Reihenfolge und die Datentypen der zu übergebenden Werte müssen mit denen im Prozedurkopf übereinstimmen. Wenn die übergebenen Werte Variablen sind, so werden sie von der Prozedur nicht verändert (sie sind lokal).

Die Liste der zu übergebenden Variablen wird Parameterliste genannt.

Übungen:

9.3.1 Schreiben Sie das Programm „Warenpreis" so um, daß kaufmännisch gerundet wird.

9.3.2 Welcher Unterschied besteht zwischen den Variablen der Parameterliste und denen des Deklarationsteils einer Prozedur?

9.3.3 Wie viele Werte müssen einer Prozedur bei ihrem Aufruf mitgegeben werden?

Unterschied

9.3.4 Schreiben Sie eine Prozedur „Bruchausgabe", die einen Bruch, der durch Zähler und Nenner gegeben ist, in folgender Form ausgibt:

$$-----1\frac{5}{7}$$

9.4 Die Prozeduren machen sich an die Variablen des Hauptprogramms heran

call-by-variable

Als letzte und komfortabelste Art der Variablenübergabe zwischen Prozeduren und Programm werden wir die Variablenübergabe vom Typ call-by-variable kennenlernen.

call-by-reference

Bei dieser Art der Übergabe werden Variablen an die Prozedur übergeben, die nach Durchlaufen der Prozedur verändert wieder zurückgegeben werden. Oft liest man in der Literatur auch den Ausdruck call-by-reference für diese Art der Übergabe.

Form

```
PROCEDURE name (VAR a,b : INTEGER; VAR c : CHAR);
  CONST ...;
  VAR   ...;

  BEGIN
  ...
  END;
```

Unterschied

Der Unterschied zu call-by-value liegt in dieser Prozedur darin, daß das reservierte Wort VAR vor denjenigen Variablen steht, die nach Durchlaufen der Prozedur verändert wieder zurückgegeben werden.

Wir wollen den uns den Unterschied an einem kleinen Programm klarmachen:

Programm

```
PROGRAM Unterschied;

  VAR x : INTEGER;

  PROCEDURE Aenderenichts (a : INTEGER);
    BEGIN
      a := a * 2;
      WRITELN (a)
    END;

  PROCEDURE Aenderewas (VAR a : INTEGER);
    BEGIN
      a := a * 2;
      WRITELN (a)
    END;

  BEGIN (* Hauptprogramm *)
    x := 5;
    WRITELN (x);
    Aenderenichts (x);
    WRITELN (x);
    Aenderewas (x);
    WRITELN (x)
  END.
```

Zeichne

Das Programm gibt folgende Zahlenreihe aus:

Ergebnis

```
 5
10
 5
10
10
```

Warum?

Analyse

5 : x hat den Wert 5 bekommen, der mit WRITELN (x) ausgegeben wird.

10 : Der Wert von x wird an die Prozedur Aenderenichts übergeben, hier mit 2 multipliziert und mit WRITELN (a) ausgegeben.

5 : Nach Durchlaufen der Prozedur hat sich die Variable x nicht geändert. Mit WRITELN (x) wird sie geschrieben.

10 : Die Variable x wird der Prozedur Aenderewas übergeben, hier verdoppelt, mit WRITELN (a) ausgegeben und der Wert von a im Hauptprogramm der Variablen x wieder zurückgegeben.

10 : Nun hat x also den Wert 10, der mit WRITELN (x) ausgegeben wird.

Fehlerquelle

Fehlerquelle: Folgender Aufruf der Prozedur Aenderewas wäre völlig falsch:

```
Aenderewas (5);
```

Denn der Prozedur muß eine Variable (call-by-variable) übergeben werden, damit der Wert dieser Variablen nach Durchlaufen der Prozedur verändert werden kann. Bei der Konstanten 5 wäre dies nicht möglich.

Mischform

Natürlich lassen sich auch Mischformen der Parameterliste erstellen:

```
PROCEDURE Zeichne (Laenge : INTEGER; VAR Winkel : INTEGER);
  BEGIN
    MOVE (Laenge);
    TURN (Winkel);
    Winkel := Winkel + 5
  END;
```

gültige Prozeduraufrufe

Seien L und W Variablen vom Typ INTEGER. Dann sind folgende Prozeduraufrufe korrekt:

```
L := 10;
W := 90;

Zeichne (L,W);          (* danach ist W=95 *)

Zeichne (10,W);
```

Falsch ist hingegen

ungültige Prozeduraufrufe

```
Zeichne (10,90);
```

da der zweite Parameter in der Prozedur als Variable erklärt ist, während der erste ein Wert ist.

Folgendes Programm soll uns den Gebrauch von Variablenübergaben an Prozeduren und zurück verdeutlichen:

Programm

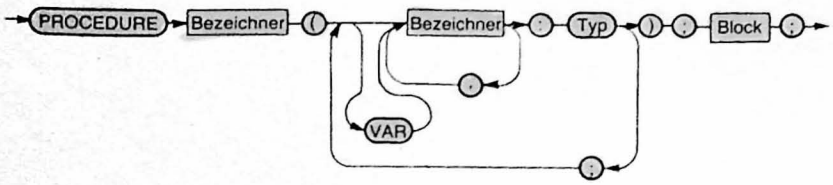
```
PROGRAM Wenigerleer;

  VAR Satz : STRING;
      Altleer, Neuleer : INTEGER;

  PROCEDURE Eingabe (VAR S : STRING);

    BEGIN
      WRITELN ('Geben Sie einen Satz ein,');
      WRITELN ('der von ueberfluessigen Leerstellen');
      WRITELN ('befreit werden soll.');
      READLN (S)
    END; (* von Eingabe *)

  PROCEDURE Leerraus (VAR S : STRING);

    VAR i, Ende : INTEGER;

    BEGIN
      Ende := LENGTH(S);
      i := 1;
      WHILE i < Ende DO BEGIN
        IF (S[i]=' ') AND (S[i+1]=' ') THEN BEGIN
          DELETE (S,i+1,1);
          Ende := Ende - 1
        END (* von IF *)
        ELSE i := i+1
      END (* von WHILE *)
    END; (* von Leerraus *)

  PROCEDURE Zaehle (S : STRING; VAR L : INTEGER);

    VAR i : INTEGER;

    BEGIN
      L := 1;
      FOR i:=1 TO LENGTH(S) DO IF S[i]=' ' THEN L:=L+1
    END; (* von Zaehle *)

  PROCEDURE Ausgabe (S : STRING; A, N : INTEGER);
    BEGIN
      WRITELN ('Der bereinigte Satz ist:');
      WRITELN (S);
      WRITELN ('Vorher enthielt er ', A:2, ' Leerstellen');
      WRITELN ('nachher ', N:2, ' Leerstellen.')
    END; (* von Ausgabe *)

BEGIN (* Hauptprogramm *)
  Eingabe (Satz);
  Zaehle (Satz, Altleer);
  Leerraus (Satz);
  Zaehle (Satz, Neuleer);
  Ausgabe (Satz, Altleer, Neuleer)
END.
```

Aufgabe: Welche Funktion hat das Programm „Wenigerleer"? Welche Prozeduren sind vom Typ call-by-variable und welche vom Typ call-by-value?

Procedur call-by-variable

Definition Prozedur vom Typ call-by-variable

Prozeduren mit Variablenübergabe vom Typ call-by-variable (oder call-by-reference)



Dieses Syntaxdiagramm ist auch für andere Prozeduren allgemein gültig, denn man kann die nicht benötigten Teile weglassen. Variablennamen, die durch das reservierte Wort VAR in der Klammer hinter dem Prozedurnamen aufgeführt werden, sind Variablen, die der Prozedur übergeben werden und nach Durchlaufen der Prozedur der entsprechenden Variablen verändert zurückgegeben werden.

Übungen:

9.4.1 Welche Pfade im Syntaxdiagramm müssen durchlaufen werden, um eine einfache Prozedur aus Kap. 9.1 zu beschreiben?

9.4.2
```
PROGRAM Demo;
VAR Global : INTEGER;

    PROCEDURE Wert (Lokal : INTEGER);
        BEGIN
            Global := 3;
            Lokal := Lokal + 2;
            Lokal := Lokal + Global
        END;

BEGIN (* Hauptprogramm *)
    Global := 0;
    Wert (Global);
    WRITELN (Global:3)
END.
```

Welches Ergebnis wird ausgegeben?

9.4.3 Ändern Sie den Prozedurkopf von 9.4.2 in

```
Wert (VAR Lokal : INTEGER);
```

und geben Sie an, welches Ergebnis nun ausgegeben wird.

9.4.4 Schreiben Sie eine Prozedur, die bei der Übergabe einer beliebigen Dezimalzahl und eines Operators vom Typ CHAR dafür sorgt, daß die Zahl abhängig vom Operator nach der zweiten Dezimalstelle abgeschnitten oder gerundet wird.

9.4.5 Schreiben Sie eine Prozedur, die bei der Übergabe einer ganzen Zahl (INTEGER) deren Fakultät (Long INTEGER) mit maximaler Stellenzahl ausgibt.

Testen Sie die Prozedur in einem entsprechenden Programm.

9.5 Zusammenfassung

Nachdem wir nun Unterprogramme (Prozeduren) kennengelernt haben und die verschiedenen Arten der Variablenübergabe beherrschen, fängt eigentlich das strukturierte Programmieren erst richtig an. Wir können nun Teile von geplanten Programmen für sich testen und später in ein übergeordnetes Programm als Prozedur einfügen. Da wir bei unseren Prozeduren frei in der Namensgebung der Variablen sind, können mehrere Leute an ein und demselben Programm (Teamwork) arbeiten, indem einzelne Gruppen Programmteile als Prozeduren erstellen. Die Gruppen sind recht unabhängig voneinander. Man muß sich vorher nur darüber verständigen, wie viele Variablen von welchen Datentypen und in welcher Reihenfolge die Prozeduren austauschen.

Die vielfältigen Möglichkeiten der Arbeit mit Prozeduren sollten viel und intensiv geübt werden.

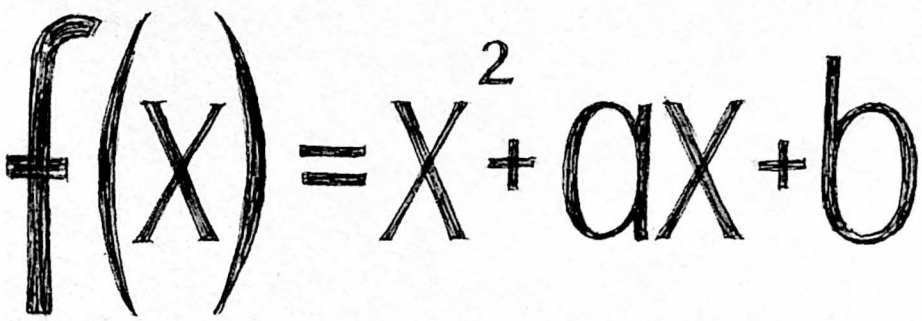
Funktionen

Kapitel 10

In diesem Kapitel wollen wir uns mit Funktionen beschäftigen. Dies sind den Prozeduren recht ähnliche Konstruktionen, die sich jedoch von den Prozeduren darin unterscheiden, daß sie genau einen Wert als Ergebnis haben; d.h., wir rufen eine Funktion mit oder ohne Variablenübergabe auf und erhalten einen Wert zugewiesen. Da es erfahrungsgemäß im Umgang mit Prozeduren und Funktionen zu Verwechselungen und daraus resultierenden Fehlern kommt, werden wir uns besonders mit den Unterschieden zwischen beiden Konstruktionen auseinandersetzen.

Nach dem Lesen dieses Kapitels

- kennen wir die Struktur und Syntax von Funktionen.
- wissen wir, wann man Prozeduren und wann Funktionen verwendet.
- kennen wir den Unterschied zwischen Prozeduren und Funktionen.



10.1 Unterschied Funktion – Prozedur

```
SIN (x)
MOVE (l)
GRAFMODE
SCREENBIT (x,y)
GOTOXY (x,y)
SQRT (x)
TURTLEANG
WRITELN
WRITELN (x)
COS (x)
```

Zwei Gruppen

Wie können wir die oben aufgeführten Anweisungen in zwei Gruppen aufteilen?
Dazu überlegen wir uns, wie die Anweisungen im Programm aufgerufen, d.h. verwendet werden (bei Unklarheiten blättern Sie einige Kapitel zurück).

So ist es z.B. nicht möglich, einfach

```
SIN (x);
```

als Programmzeile zu schreiben. Vielmehr muß der Wert des Sinus einer Variablen zugeordnet werden. Daher können wir schreiben:

```
y := SIN (x);
```

Andererseits dürfen wir einfach

```
WRITELN;
```

als Programmzeile schreiben.

Wertzuweisung

Wenn wir die oben aufgeführten Anweisungen danach untersuchen, ob sie allein oder im Zusammenhang mit einer Wertzuweisung zu benutzen sind, dann ergeben sich folgende Gruppen:

Gruppe 1:	Gruppe 2:
SIN (x)	MOVE (l)
SCREENBIT (x,y)	GRAFMODE
SQRT (x)	GOTOXY (x,y)
TURTLEANG	WRITELN
COS (x)	WRITELN (x)

Funktionen

Anweisungen der Gruppe 1 haben immer ein Ergebnis, das z.B. einer Variablen zugeordnet oder ausgegeben werden kann.

Diese Anweisungen werden FUNKTIONEN genannt!

SIN (x)	hat den Sinus von x als Datentyp REAL zum Ergebnis.
SCREENBIT (x,y)	hat das Ergebnis TRUE (vom Datentyp BOOLEAN), wenn der Punkt (x,y) nicht schwarz ist.
SQRT (x)	hat die zweite Wurzel von x als Datentyp REAL zum Ergebnis.
TURTLEANG	hat den aktuellen Winkel des Turtle als Datentyp INTEGER zum Ergebnis.
COS (x)	hat den Kosinus von x als Datentyp REAL zum Ergebnis.

Prozeduren

Anweisungen der Gruppe 2 stehen für sich allein.

Diese Anweisungen werden PROZEDUREN genannt.

MOVE (l)	bewegt den Turtle um l weiter.
GRAFMODE	schaltet auf Grafik um.
GOTOXY (x,y)	bringt den Cursor an die Stelle (x,y) des Textbildschirms.

Beispiel Funktion

WRITELN schreibt einen Zeilenvorschub.

WRITELN (x) schreibt die Variable x mit einem Zeilenvorschub.

Sowohl Funktionen als auch Prozeduren führen irgendwelche Anweisungen aus. Beiden können Variablen mitgegeben werden.

Funktionen jedoch haben nach der Ausführung der Anweisungen genau einen Wert, der einer Variablen zugewiesen oder ausgegeben werden kann.

Beispiel

```
FUNCTION name (x : REAL) : INTEGER;
  CONST. . . ;
  VAR    . . . ;
  BEGIN
     . . .
     name := . . . ;
     . . .
  END;
```

Die Funktion name bekommt einen Wert vom Datentyp REAL mitgegeben und hat nach Ausführung ein Ergebnis vom Datentyp INTEGER. Im Anweisungsteil wird dem Funktionsnamen ein Funktionswert zugewiesen.

Diesen Wert repräsentiert die Funktion von nun an, wenn sie mit ihrem Namen (und entsprechenden Parametern) aufgerufen wird.

Mit einer Variablen a vom Typ INTEGER könnte der Funktionsaufruf z.B. so aussehen:

```
a := name (4.2);
```

oder

```
WRITELN (name (4.2):3);
```

Achtung: Rechts vom Zuweisungszeichen *muß* die Funktion mit Parametern aufgerufen werden (sofern sie welche hat).

Zuweisungsbeispiele

`name := name +2;`	falsch
`name := 15;`	richtig
`a := name;`	falsch
`a := name ('text');`	falsch
`a := name (3.74);`	richtig
`a := name (r);`	richtig, wenn r vom Datentyp REAL ist.

Aufgabe (Potenz)

Wir wollen nun eine kleine Funktion selbst schreiben. In Pascal existiert keine Standardfunktion zum Potenzieren. x^a läßt sich also nicht ohne Umwege berechnen.

Daher schreiben wir uns die entsprechende Funktion Potenzieren selbst. Um die Sache mathematisch nicht zu kompliziert zu machen, beschränken wir uns auf ganzzahlige Exponenten. Außerdem verzichten wir auf eine Überprüfung, ob der Wertbereich überschritten ist. Fortgeschrittene und interessierte Leser können die Funktion entsprechend erweitern.

Funktion

```
FUNCTION Potenz (Basis : REAL; Exponent : INTEGER) : REAL;
  VAR i : INTEGER;
      p : REAL;
      positiv : BOOLEAN;

  BEGIN
    p := 1;
    positiv := (Exponent = ABS (Exponent));
    Exponent := ABS (Exponent);
    FOR i := 1 TO Exponent DO p := p * Basis;
    IF positiv THEN Potenz := p
               ELSE Potenz := 1 / p
  END; (* von Potenz *)
```

Hinweise: Die Funktion Potenz wird aufgerufen mit zwei Werten, von denen der erste vom Typ REAL und der zweite vom Typ INTEGER sein muß. Das Ergebnis ist vom Typ REAL.

Man ist im Anweisungsteil möglicherweise geneigt, ohne die Hilfsvariable p zu arbeiten und zu schreiben:

```
FOR i:=1 TO Exponent DO Potenz := Potenz * Basis;      (* falsch! *)
```

Dies ist falsch, da auf der rechten Seite des Zuweisungszeichens die Funktion mit Parametern aufgerufen werden muß.

(Wenn eine Funktion sich in ihrem Anweisungsteil selbst aufruft, so entsteht eine Rekursion; siehe Kap. 11.)

Unsere Funktion Potenz sollten wir nun in einem kleinen Programm testen, um zu sehen, ob wir sie korrekt geschrieben haben:

Programm

```
PROGRAM Funtest;

  VAR a, x : REAL;
      b : INTEGER;
      Frage : CHAR;

  FUNCTION Potenz (Basis : REAL; Exponent : INTEGER) : REAL;
    VAR i : INTEGER;
        p : REAL;
        positiv : BOOLEAN;

    BEGIN
      p := 1;
      positiv := (Exponent = ABS (Exponent));
      Exponent := ABS (Exponent);
      FOR i := 1 TO Exponent DO p := p * Basis;
      IF positiv THEN Potenz := p
                 ELSE Potenz := 1 / p
    END; (* von Potenz *)

  BEGIN (* Hauptprogramm *)
    WRITELN ('Potenzrechnung: ');
    REPEAT
      WRITE ('Geben Sie die Basis ein: ');
      READLN (a);
      WRITE ('Geben Sie den Exponenten ein: ');
      READLN (b);
      x := Potenz (a,b);
      WRITELN (a:5:2 ,' hoch ', b:2 ,' = ', x:7:2);
      WRITELN;
      WRITE ('Noch eine Potenz (J/N) ? ');
      READ (Frage);
      WRITELN
    UNTIL (Frage='N') OR (Frage='n')
  END.
```

FUNCTION

Wichtige Anmerkungen:

Einschränkung in UCSD

- In UCSD-Pascal (insbesondere beim APPLE-Pascal) gilt als Einschränkung, daß Funktionen nur einfache Datentypen als Parameter und als Ergebnis haben dürfen.

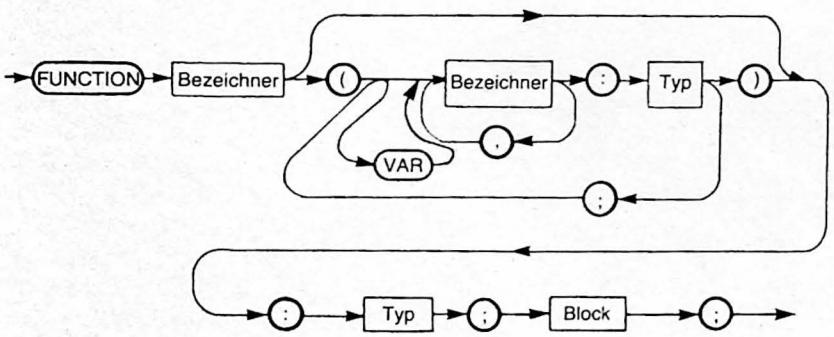
  Es sind also INTEGER, REAL, CHAR, BOOLEAN und in der TYPE-Deklaration (siehe Band 2) definierte Aufzählungs- oder Unterbereichstypen erlaubt.

Variablenübergabe

- Wie bei Prozeduren können Funktionen vom Typ call-by-value (unser Beispiel) oder call-by-variable sein. Im letzten Fall hat die Funktion ein Ergebnis und ändert zusätzlich die an sie übergebenen Variablen.

Definition FUNCTION

Funktionen:




```
FUNCTION name (Parameterliste) : Ergebnistyp;
   CONST. . . ;
   VAR    . . . ;
   BEGIN
      . . .
      name := . . .;
      . . .
   END;
```

Eine Funktion hat im Gegensatz zu einer Prozedur stets ein Ergebnis. Funktionen können mit oder ohne Parameter aufgerufen werden. Die Parameter können mit call-by-value und call-by-variable übergeben werden. Im Anweisungsteil der Funktion wird dem Funktionsnamen (links vom Zuweisungszeichen) ein Ergebnis vom Ergebnistyp zugewiesen. Beim Funktionsaufruf (rechts vom Zuweisungszeichen) müssen die Parameter der Funktion übergeben werden.

Die Parameter und das Ergebnis müssen einfache Datentypen sein.

Funktionen können wie Prozeduren geschachtelt werden. Bei der Reihenfolge gelten dieselben Regeln wie bei Prozeduren: Eine aufgerufene Funktion/Prozedur muß im Text vor der aufrufenden Funktion/Prozedur stehen.

Übungen:

10.1.1 READ(LN) und WRITE(LN) passen, genau betrachtet, nicht so recht in das Prozedur-/Funktionskonzept. Warum?

10.1.2 Muß eine Funktion eine Parameterliste haben?

10.1.3 Muß eine Funktion ein Ergebnis haben?

10.1.4 Schreiben Sie eine Funktion „Ende", die das Ergebnis TRUE hat, wenn die Leertaste gedrückt wird; andernfals FALSE.

10.2 Übungen zu Funktionen

Wir wollen nun ein paar Funktionen erstellen, um die Struktur dieser Konstruktionen besser kennenzulernen und zu üben.

Minimum *Das Minimum zweier ganzer Zahlen*

Die Funktion Min ermittelt das Minimum zweier Zahlen vom Datentyp INTEGER. Das Ergebnis ist natürlich ebenfalls vom Typ INTEGER.

```
FUNCTION Min (a,b : INTEGER) : INTEGER;
  BEGIN
    IF a < b THEN Min := a
             ELSE Min := b
  END; (* von Min *)
```

ggT *Größter gemeinsamer Teiler ggT*

Die Funktion ggT soll den größten gemeinsamen Teiler zweier Zahlen vom Datentyp INTEGER ermitteln. Das Ergebnis ist selbstverständlich ebenfalls vom Datentyp INTEGER.

Dazu zählen wir eine Variable namens Teiler von 1 bis zum Minimum der beiden Zahlen, denn der gemeinsame Teiler kann höchstens so groß sein wie die kleinere Zahl. Immer dann, wenn sich beide Zahlen durch Teiler teilen lassen, ist ggT gleich diesem Teiler.

Für das Minimum benutzen wir gleich die schon geschriebene Funktion.

```
FUNCTION ggT (a,b : INTEGER) : INTEGER;
  VAR Teiler : INTEGER;
  BEGIN
    FOR Teiler := 1 TO Min(a,b) DO
      IF a MOD Teiler = 0 THEN IF b MOD Teiler = 0 THEN ggT := Teiler
  END; (* von ggT *)
```

Primzahltest *Prüfen einer Zahl als Primzahl*

Die Funktion Prim ermittelt, ob eine Zahl vom Datentyp INTEGER eine Primzahl ist oder nicht. Das Ergebnis ist vom Datentyp BOOLEAN, also wahr oder falsch.

Eine Zahl, die kleiner als 2 ist, ist keine Primzahl. Dann wird geprüft,

Prim / Weiter / Zahlenzauber

ob die Zahl gleich 2 ist, dann ist sie Primzahl, oder ob sie durch eine folgende ungerade Zahl (3, 5, 7, 9, ...) teilbar ist. Gerade Zahlen > 2 können sowieso keine Primzahlen sein. Am besten wäre, wir würden versuchen, durch alle ungeraden Zahlen bis ROUND (SQRT (Zahl)) zu teilen. Dies bringt jedoch beim APPLE die Notwendigkeit mit sich, im Hauptprogramm USES TRANSCEND zu erklären. Um auf USES TRANSCEND zu verzichten (damit läßt sich die Funktion universeller einsetzen), lassen wir die Teiler bis zur größten ungeraden Zahl, die kleiner als die eingegebene Zahl ist, laufen.

```
FUNCTION Prim (Zahl : INTEGER) : BOOLEAN;
  VAR Teiler : INTEGER;
      Test : BOOLEAN;
  BEGIN
    Teiler := 3;
    Test := TRUE;
    IF Zahl < 2 THEN Test := FALSE
      ELSE IF Zahl > 2 THEN
        REPEAT
          IF Zahl MOD Teiler = 0 THEN Test := FALSE;
          Teiler := Teiler + 2
        UNTIL Teiler >= Zahl;
    Prim := Test
  END; (* von Prim *)
```

Weitermachen (J/N)? ***Weiter?***

Die Funktion Weiter fragt den Benutzer, ob er im Programm weiter fortfahren möchte oder nicht. Das Ergebnis ist ebenfalls wieder vom Datentyp BOOLEAN. Die Funktion eignet sich für Programme, in denen oft nach einer Fortführung des Programms gefragt wird.
In Schleifen könnte dann die Funktion zum Einsatz kommen mit: ***Anwendung***

```
WHILE Weiter DO . . . ;
```

oder mit:

```
REPEAT
  . . .
UNTIL NOT Weiter;
```

```
FUNCTION Weiter : BOOLEAN;
  VAR ch : CHAR;
  BEGIN
    WRITE ('Wollen Sie weitermachen (J/N)? ');
    READ (ch);
    Weiter := (ch='J') OR (ch='j')
  END; (* von Weiter *)
```

Ein Programm

Damit wir die oben erstellten Funktionen testen können, wollen wir sie in ein kleines Programm integrieren und dieses ausführen:

Programm

```
PROGRAM Zahlenzauber;

  VAR Zahl1, Zahl2 : INTEGER;
      Wort : STRING;
```

(Fortsetzung des Programms)

```
FUNCTION Min (a,b : INTEGER) : INTEGER;
  BEGIN
    IF a < b THEN Min := a
             ELSE Min := b
  END; (* von Min *)

FUNCTION ggT (a,b : INTEGER) : INTEGER;
  VAR Teiler : INTEGER;

  BEGIN
    FOR Teiler := 1 TO Min(a,b) DO
      IF a MOD Teiler = 0 THEN
        IF b MOD Teiler = 0 THEN ggT := Teiler
  END; (* von ggT *)

FUNCTION Prim (Zahl : INTEGER) : BOOLEAN;
  VAR Teiler : INTEGER;
      Test : BOOLEAN;
  BEGIN
    Teiler := 3;
    Test := TRUE;
    IF Zahl < 2 THEN Test := FALSE
      ELSE IF Zahl > 2 THEN
        REPEAT
          IF Zahl MOD Teiler = 0 THEN Test := FALSE;
          Teiler := Teiler + 2
        UNTIL Teiler >= Zahl;
    Prim := Test
  END; (* von Prim *)

FUNCTION Weiter : BOOLEAN;
  VAR ch : CHAR;
  BEGIN
    WRITE ('Wollen Sie weitermachen (J/N)? ');
    READ (ch);
    Weiter := (ch='J') OR (ch='j')
  END; (* von Weiter *)

BEGIN (* Hauptprogramm *)
  REPEAT
    WRITELN ('Geben Sie zwei ganze Zahlen ein:');
    WRITE ('1. Zahl: ');
    READLN (Zahl1);
    WRITE ('2. Zahl: ');
    READLN (Zahl2);
    WRITELN ('Die kleinere Zahl ist ', Min(Zahl1,Zahl2):3);
    WRITELN ('Der groesste gemeinsame Teiler ist:');
    WRITELN ('ggT(',Zahl1:3,',',Zahl2:3,')=',ggT(Zahl1,Zahl2):3);
    IF Prim (Zahl1) THEN Wort:='eine' ELSE Wort:='keine';
    WRITELN (Zahl1:3,' ist ',Wort,' Primzahl');
    IF Prim (Zahl2) THEN Wort:='eine' ELSE Wort:='keine';
    WRITELN (Zahl2:3,' ist ',Wort,' Primzahl')
  UNTIL NOT Weiter
END.
```

Übungen:

10.2.1 Schreiben Sie eine Funktion, die das kleinste gemeinsame Vielfache (kgV) von 2 Zahlen berechnet.

10.2.2 Schreiben Sie eine Funktion „Lieszeichen“ mit folgendem Kopf:

FUNCTION Lieszeichen (Anfang, Ende : CHAR) : CHAR;

zu folgender Aufgabe:

Der Funktion werden die Anfangs- und End-Ordnungsnummern eines Bereichs von ASCII-Zeichen übergeben. Mit

READ(KEYBOARD, ...) wird ein Zeichen von der Tastatur gelesen und nur dann akzeptiert und als Funktionsergebnis ausgegeben, wenn das Zeichen aus dem Bereich ist. Andernfalls wird ein Piepton (CHR(7)) ausgegeben und die Eingabe wiederholt.

Im Programm ist dies eine sehr nützliche Funktion. Soll z.B. bei einer Eingabe nur ein Großbuchstabe erlaubt sein, so kann folgender Aufruf erfolgen:

```
ch := Lieszeichen ('A','Z');
```

10.3 Zusammenfassung

Mit den Funktionen haben wir Konstruktionen kennengelernt, die durch den einfachen Aufruf ihres Namens (verbunden mit möglichen Parametern) ein Ergebnis liefern.

Zwar könnten wir mit Prozeduren auch Funktionen ersetzen, dies würde jedoch zu recht unübersichtlichen Konstruktionen führen. Prozeduren können ja Variablen, die in ihrer Parameterliste stehen, verändern. Eindeutiger ist da die Funktion, die eine Reihe von Anweisungen ausführt und schließlich zu einem Ergebnis führt.

Rekursionen

Kapitel 11

Ein entscheidender Vorteil einer Programmiersprache, die lokale und globale Variablen kennt, ist die einfache Möglichkeit, Rekursionen zu konstruieren.

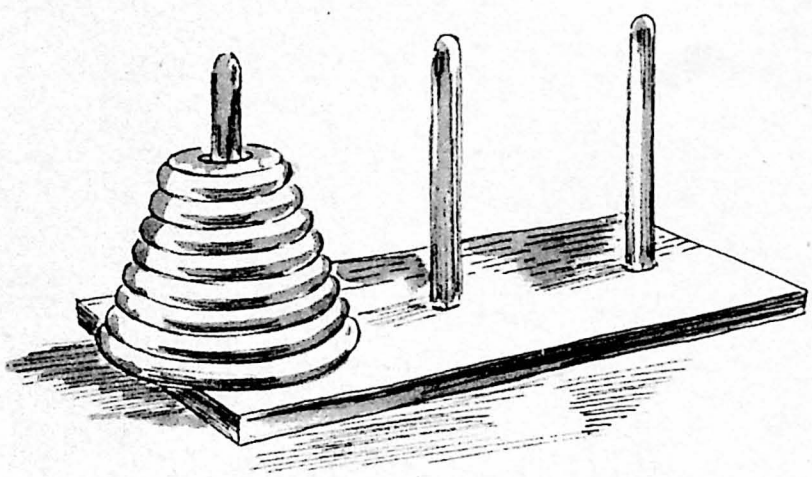
Von einer Rekursion sprechen wir, wenn eine Prozedur oder Funktion sich selbst aufruft. Dies ist in anderen Programmiersprachen auch möglich (wer BASIC kennt, sollte nach dem Durcharbeiten dieses Kapitels einmal Rekursionen in BASIC probieren), jedoch können in Pascal (und anderen blockorientierten Sprachen wie LOGO) Variablen mit übergeben werden.

Häufig werden Probleme aus der Mathematik mit Rekursionen beschrieben, z.B. bei Bildungsvorschriften für Folgen und Reihen. Hier ist es dann sehr einfach, die Rekursionsvorschrift in Pascal umzuschreiben. Jedoch auch in anderen Bereichen (Grafik, Textverarbeitung und später bei sogenannten Baumstrukturen) eignen sich Rekursionen vorzüglich für die Programmierung dieser Probleme.

Allerdings soll auch darauf hingewiesen werden, wann Rekursionen besser zu vermeiden sind.

Nach dem Lesen dieses Kapitels

- wissen wir, was eine Rekursion ist.
- können wir rekursive Prozeduren und Funktionen programmieren.
- kennen wir mögliche Fehlerquellen bei Rekursionen und können sie vermeiden.
- kennen wir die Probleme mit der Rechenzeit und dem Stapelspeicher bei der Verwendung von Rekursionen.
- wissen wir, daß man manche Rekursionen besser vermeidet.



11.1 Was ist eine Rekursion?

Optische Rekursion

In der Zeitung sehen wir eine Werbeanzeige einer Waschmittelfirma:
Eine Frau hat eine Waschmittelpackung in der Hand.
Auf der Waschmittelpackung hat eine Frau eine Waschmittelpackung in der Hand.
Auf der Waschmittelpackung hat eine Frau eine Waschmittelpackung in der Hand.
Auf der Waschmittelpackung ...
...

Sind unendlich viele Frauen mit Waschmittelpackungen (immer kleiner werdend) auf dem Bild? Natürlich nicht. Denn irgendwann ist das Auflösungsvermögen der Druckmaschine erschöpft, und wir sehen nur noch einen Punkt.

Rekursion

Unser Bild stellt eine Rekursion dar.

Programm

Wir wollen uns einer programmierten Rekursion zuwenden:

```
PROGRAM Was_macht_das; (* Idee: Pascal-Kurs der FU-Hagen *)

  PROCEDURE Zeichen;

    VAR ch : CHAR;

    BEGIN
      READ (ch);
      IF ch <> ' ' THEN Zeichen;
      WRITE (ch)
    END; (* von Zeichen *)

    BEGIN (* Hauptprogramm *)
  Zeichen
    END.
```

Analyse

Was macht das Programm?

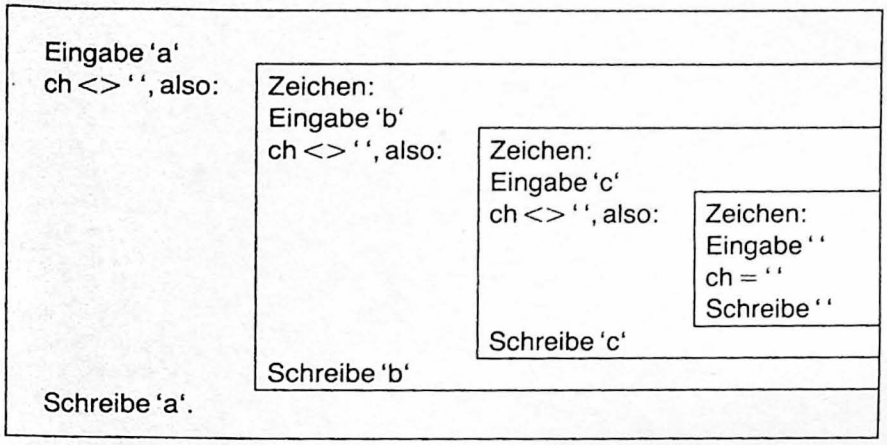
Zur Übung sollte jetzt das Buch beiseite gelegt werden.

Daß es sich hier um eine Rekursion handelt, erkennen wir an dem Aufruf der Prozedur Zeichen innerhalb der Prozedur Zeichen. Die Prozedur ruft sich also selbst auf.

rekursive Prozedur

Nehmen wir an, wir geben die Buchstaben 'abc' und danach eine Leerstelle ein.

Das Hauptprogramm ruft die Prozedur Zeichen auf:



Die Eingabe eines Zeichens wird also so lange fortgesetzt, bis das Leerzeichen eingegeben wird. Dann werden alle Zeichen rückwärts wieder ausgegeben.

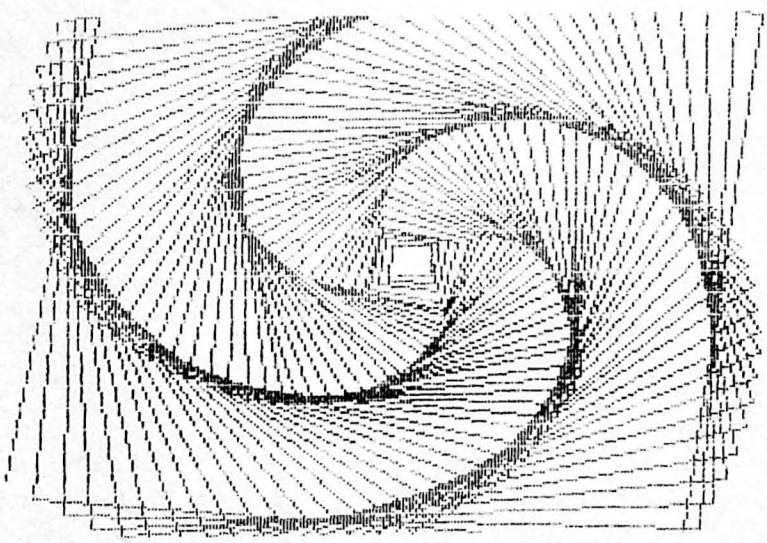
Lokale Variablen (immer neu)

In dieser Art ist die Rekursion natürlich nur möglich, wenn bei jedem Aufruf der Prozedur Zeichen eine neue Variable ch bereitgestellt wird. Dies ist dadurch gewährleistet, daß ch eine lokale Variable ist und somit nur innerhalb der Prozedur definiert ist. Obwohl die Variablen gleiche Namen haben, sind sie doch unterschiedlich.

Selbstaufruf

→ Merke: Wenn eine Prozedur oder Funktion sich selbst aufruft (rekursiv aufruft), dann werden nach der Ausführung der Prozedur/Funktion die restlichen Anweisungen noch abgearbeitet.

Ein weiteres Beispiel einer rekursiven Prozedur, diesmal unter Zuhilfenahme der Grafik:

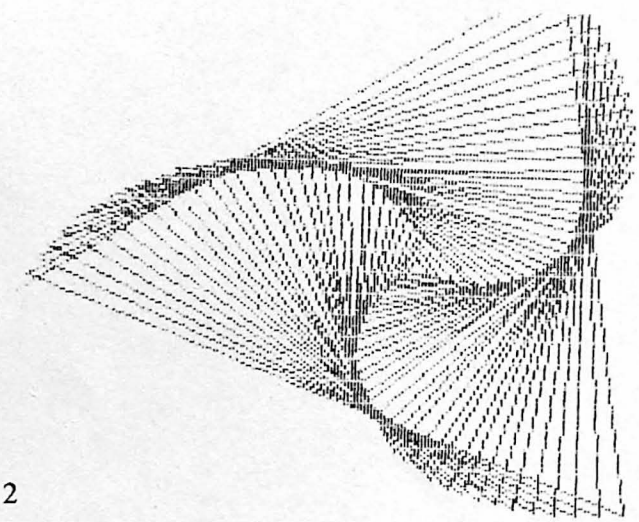


10, 89, 1

Spirale

Programm

```
PROGRAM Spirale;

  USES TURTLEGRAPHICS;

  VAR L, W, Z : INTEGER;

  PROCEDURE Seite (Laenge, Winkel, Zuwachs : INTEGER);
    BEGIN
      IF Laenge < 200 THEN BEGIN
        MOVE (Laenge);
        TURN (Winkel);
        Laenge := Laenge + Zuwachs;
        Seite (Laenge, Winkel, Zuwachs)
      END (* von IF *)
    END; (* von Seite *)

  BEGIN (* Hauptprogramm *)
    WRITELN ('Spiralen...');
    WRITE ('Eingabe der Anfangsseite: ');
    READLN (L);
    WRITE ('Eingabe der Winkelaenderung: ');
    READLN (W);
    WRITE ('Eingabe des Seitenzuwachses: ');
    READLN (Z);
    INITTURTLE;
    VIEWPORT (0, 279, 0, 191);
    PENCOLOR (WHITE);
    Seite ( L, W, Z );
    READLN (* zum Anhalten *)
  END.
```



5, 121, 2

Analyse Die Prozedur Seite ruft sich in diesem Programm stets selbst wieder auf, wobei die Seitenlänge jeweils um den Seitenzuwachs vergrößert und der Änderungswinkel unverändert übergeben wird.

Aufgabe: Ändern Sie die Prozedur, indem die Anweisung IF Laenge<200 THEN herausgenommen wird. Welchen Effekt hat dies?

Rekursionsabbruch Nun, die Prozedur ruft sich dann unendlich lange selbst auf. Tatsächlich hört das Programm allerdings irgendwann ziemlich unschön auf, weil die Rechenzeit überschritten wurde, der Stapelspeicher (Stack) übergelaufen ist oder weil der Rechner die Grafik mit den immer größer werdenden Seiten nicht mehr bewältigt.

Fakultät

Fehlerquelle: Es ist stets darauf zu achten, daß eine Rekursion eine Abbruchbedingung hat. Wenn also eine Prozedur/Funktion sich selbst aufruft, so sollte der Aufruf von einer Bedingung abhängen, deren Wahrheitsgehalt sich irgendwann zu FALSE ändert.

Fehler bei Rekursionen

Wir wollen uns nun noch ein kleines Beispiel für eine rekursive Funktion ansehen. Aus der Mathematik ist sicher die sogenannte Fakultät bekannt. Es ist z.B. $5!=1*2*3*4*5$.

Aufgabe (Fakultät)

Oft wird das Bildungsgesetz für eine beliebige Fakultät jedoch so dargestellt:

$$1! = 1 \qquad \text{und} \qquad n! = (n-1)! * n$$

Dies ist eine rekursive Beschreibung der Fakultät, denn n! wird berechnet durch das Produkt aus n und der Fakultät des Vorgängers von n. Wir tun so, als wenn wir $(n-1)!$ berechnen könnten. Können wir auch, denn $(n-1)! = (n-2)! * (n-1)$ usw.

Jedoch wäre dies ein unendlicher Prozeß, gäbe es nicht die Abbruchbedingung (oft auch Rekursionsanfang genannt), die lautet: $1!=1$.

Als Pascal-Funktion liest sich das so:

Funktion (rekursiv)

```
FUNCTION Fak (n : INTEGER) : INTEGER;
  BEGIN
    IF n=1 THEN Fak := 1
           ELSE Fak := Fak(n-1) * n
  END; (* von Fak *)
```

So einfach ist das. Wir haben nur die Rekursionsvorschrift von n! in Pascal-Anweisungen umgeschrieben.

Hinweis: Da Fakultäten schnell sehr groß werden, ist MAXINT rasch überschritten. Dies ist bei der Funktion nicht berücksichtigt, um nicht vom Thema abzulenken.

Aufgabe: Schreiben Sie ein kleines Programm, um die Funktion zu testen.

Hätten wir ohne die Möglichkeit der Rekursion die Funktion Fak nicht schreiben können? Doch, aber der Vorteil der Rekursion liegt in der Einfachheit und Übersichtlichkeit. Daß Rekursionen auch erhebliche Nachteile haben können, zeigt der nächste Paragraph.

Jede Rekursion läßt sich jedoch durch eine Iteration (Nacheinanderausführung von Anweisungen) ersetzen. Am Beispiel der Fakultät sähe das folgendermaßen aus:

Iteration

Funktion (iterativ)

```
FUNCTION Fak (n : INTEGER) : INTEGER;
  Var i, Hilf : INTEGER;
  BEGIN
    Hilf := 1;
    IF n=1 THEN Hilf := 1
           ELSE FOR i:=1 TO n DO Hilf := Hilf * i;
    Fak := Hilf
  END; (* von Fak *)
```

Beachte: Ohne die Hilfsvariable Hilf kämen wir nicht aus, da die Zuweisung Fak := Fak * i nicht erlaubt wäre. Warum?

Aufgabe: Testen Sie auch diese Funktion in einem Programm!

Definition Rekursion

Rekursion:

Von einer Rekursion sprechen wir, wenn sich eine Prozedur oder Funktion selbst aufruft. Dabei sind Konstanten und Variablen beim erneuten Aufruf der Prozedur/Funktion nicht mit den gleichnamigen der aufrufenden Prozedur/Funktion identisch.

Eine Rekursion muß stets eine Abbruchbedingung haben, die auch erreicht wird. Andernfalls käme es zu einer unendlichen Rekursion.

Direkte und indirekte Rekursion

Hinweis: Wenn eine Prozedur/Funktion sich explizit selbst aufruft, sprechen wir von einer direkten Rekursion. Eine indirekte Rekursion dagegen liegt vor, wenn eine Prozedur/Funktion eine andere aufruft, die dann ihrerseits einen Aufruf der ersten Prozedur/Funktion enthält.

Übungen:

11.1.1 Schreiben Sie eine Funktion zum Potenzieren ganzer Zahlen, die rekursiv über eine Multiplikation läuft, die selbst rekursiv über eine Addition und diese über die Nachfolger-Funktion (SUCC) erstellt wird.

11.1.2 Überlegen Sie sich jeweils ein Beispiel für eine direkte und eine indirekte Rekursion.

11.1.3 Schreiben Sie ein Programm mit rekursiven Algorithmen, das einen Taschenrechner mit 4 Grundrechenarten und Klammern simuliert. D.h., zugelassene Zeichen sind:

0 1 2 3 4 5 6 7 8 9 + − * / ( ) = . und C zum Löschen der letzten Zahl.

11.2 Probleme der Rechenzeit und des Speicherplatzes mit Rekursionen

```
fib(0) =  0
fib(1) =  1
fib(2) =  1
fib(3) =  2
fib(4) =  3
fib(5) =  5
fib(6) =  8
fib(7) =13
fib(8) =21
fib(9) =34
....
```

Beispiel Fibonacci-Zahlen

Die Zahlen dieser Reihe nennt man Fibonacci-Zahlen. Sie werden gebildet, indem jeweils die letzte und vorletzte Fibonacci-Zahl zusammenaddiert werden.

Fibonacci-Zahlen

Das mathematische Bildungsgesetz sieht folgendermaßen aus:

$$\begin{aligned} fib(0) &= 0 \\ fib(1) &= 1 \\ fib(n) &= fib(n-1) + fib(n-2) \end{aligned}$$

Na, also. Wieder eine Rekursion, wie wir sofort erkannt haben. Da wir ja nun schon einige Übung im Programmieren haben, können wir sicher sofort die dazugehörige Pascal-Funktion schreiben:

Funktion (rekursiv)

```
FUNCTION fib (n : INTEGER) : INTEGER;
  BEGIN
    IF n<2 THEN fib := n
           ELSE fib := fib(n-1) + fib(n-2);
  END; (* von fib *)
```

Testen wir unsere Funktion in einem Programm:

Programm (rekursiv)

```
PROGRAM Rekufib;
  VAR a : INTEGER;

  FUNCTION fib (n : INTEGER) : INTEGER;
    BEGIN
      IF n<2 THEN fib := n
             ELSE fib := fib(n-1) + fib(n-2);
    END; (* von fib *)

  BEGIN (* Hauptprogramm *)
    WRITE ('Eingabe einer Zahl: ');
    READLN (a);
    WRITELN ('fib(', a:3, ')= ', fib(a):5)
  END.
```

Alle Zahlen unserer oben aufgeführten Fibonacci-Reihe berechnet das Programm einwandfrei (testen Sie das!).

Geben wir nun einmal eine etwas größere Zahl ein: 22.

Rechenzeit

Ich dachte immer, Computer seien schnell...
Aber es dauert beim APPLE etwa 1 Min. (in Worten: 60 Sekunden), bis wir das Ergebnis fib(22)=17711 erhalten.

Zum Vergleich wollen wir die Funktion fib iterativ schreiben und anstelle der rekursiven Funktion in unser Programm aufnehmen:

Programm (iterativ)

```
PROGRAM Iterfib;
  VAR a : INTEGER;

  FUNCTION fib (n : INTEGER) : INTEGER;
    VAR i, Hilf, Letzt, Vorletzt : INTEGER;
    BEGIN
      Letzt := 1;
      Vorletzt := 0;
```

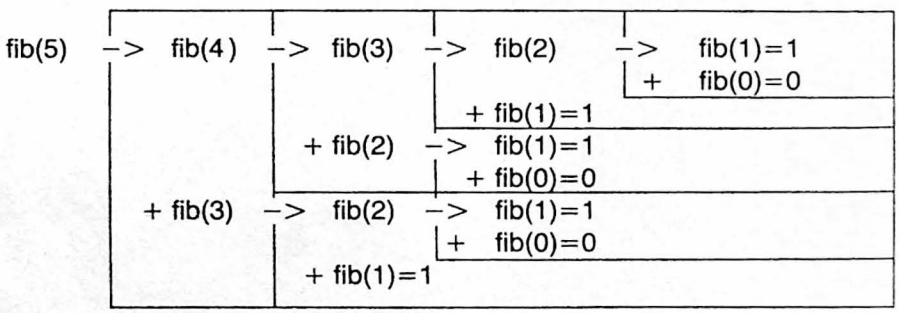
(Fortsetzung des Programms)

```
      IF n<2 THEN Hilf := n
      ELSE FOR i:=2 TO n DO BEGIN
             Hilf := Letzt + Vorletzt;
             Vorletzt := Letzt;
             Letzt := Hilf
           END; (* von FOR *)
      fib := Hilf
    END; (* von fib *)

BEGIN (* Hauptprogramm *)
  WRITE ('Eingabe einer Zahl: ');
  READLN (a);
  WRITELN ('fib(', a:3, ')=', fib(a):5)
END.
```

Bei dieser Version des Programms stoppen wir eine Zeit unter einer Sekunde für fib(22).

Unterschied Woher dieser gewaltige Unterschied?

Analyse Machen wir uns einmal deutlich, wie fib(5) rekursiv berechnet wird:

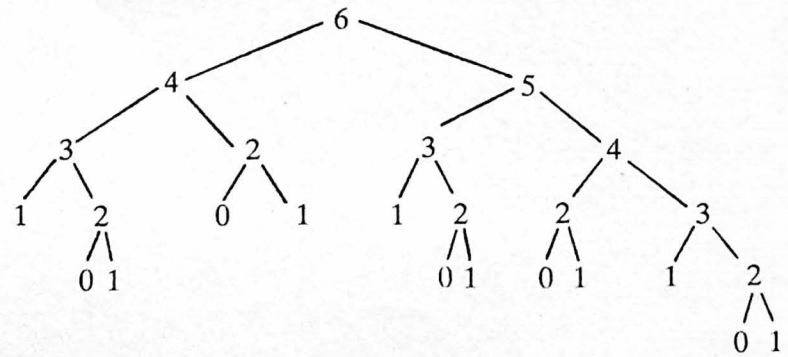


Addieren wir alle Endwerte (1) bei den Rekursionsabbrüchen fib(1) und fib(0), so erhalten wir als Ergebnis 5.

Wir erkennen, daß für n=5 die Funktion 15mal aufgerufen wird.

Eine andere Darstellung der Aufrufe der rekursiven Funktion ist ein Baum (hier für fib(6)=8):

Baum



Tiefe

Hier wird die Funktion schon 25mal aufgerufen.

Stapel

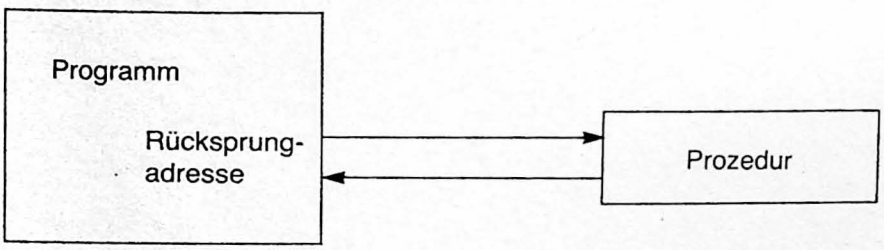
Wir sehen also, daß die Zahl der Funktionsaufrufe explosionsartig ansteigt, da sich die Funktion jeweils zweimal selbst aufruft. Daher kommt die enorme Rechenzeit bei der Rekursion. Für die Iteration wird nur einmal die Funktion aufgerufen und eine Schleife von 2 bis n abgearbeitet.

Ein anderes Problem bei Rekursionen spielt ebenfalls eine große Rolle: der Stapelspeicher (Stack).

Stack (Stapelzeichner)

Wenn eine Funktion/Prozedur aufgerufen wird, muß der Rechner sich die Rücksprungadresse merken, denn nach dem Abarbeiten der Funktion/Prozedur soll es ja an der gleichen Stelle weitergehen.

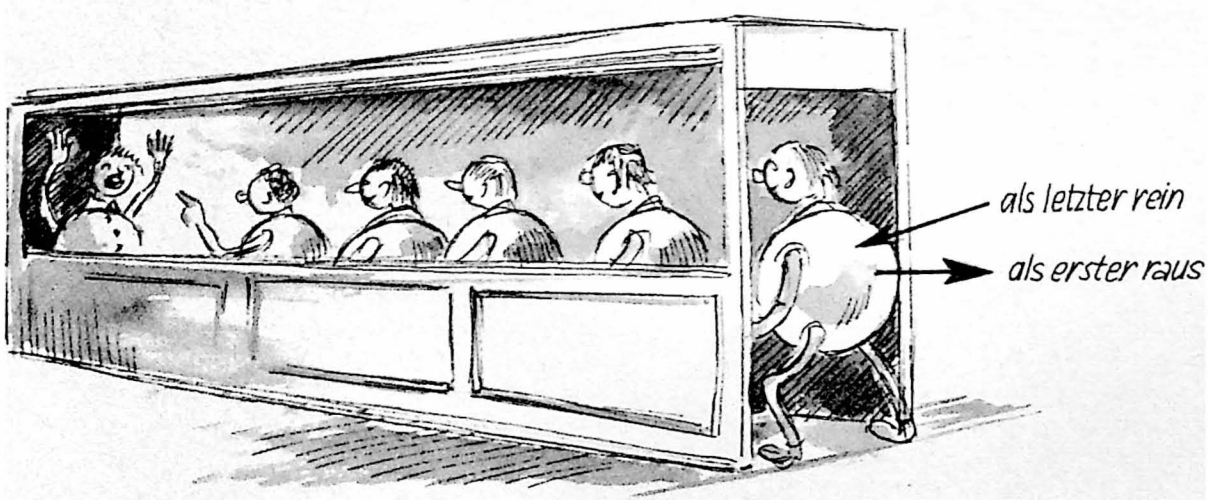
Rücksprung-adressen



Außerdem müssen jeweils die lokalen Variablen der Funktion/Prozedur abgelegt werden.

LIFO-Stack

All dies geschieht im sogenannten Stack (oder Stapelspeicher oder Kellerspeicher).
Es handelt sich hier um einen LIFO-Stack (Last-In-First-Out). Diesen Speicherbereich können wir uns vorstellen wie einen Parkgroschenspender:



Der letzte Groschen, der in den Spender eingelegt wird, wird auch als erster wieder herausgenommen.

Für Funktionen/Prozeduren heißt das, die zuletzt aufgerufene Funktion/Prozedur kehrt als erste wieder zu ihrer Ausgangsposition zurück.

Überlastung des Stack

Bei Rekursionen kann es nun (abhängig von der Größe des Stack unseres Rechners) leicht zu einer Überlastung des Stack kommen, wenn die Funktion/Prozedur sich zu häufig aufruft (zu große Rekursionstiefe).

In unserem vorigen Beispiel der Fakultät (Kap 11.1) spielten diese Probleme kaum eine Rolle, da die Funktion sich selbst jeweils nur einmal aufruft, d.h., für n! wird die Funktion n-mal abgearbeitet.

Regeln

Probleme bei Rekursionen:

Bei zu großer Rekursionstiefe (zu häufigem Selbstaufruf der Funktion/Prozedur) erhöht sich oft die Rechenzeit erheblich, und es kann zum Überlaufen des Stapelspeichers (Stack) kommen, was einen Programmabbruch mit sich bringen kann.

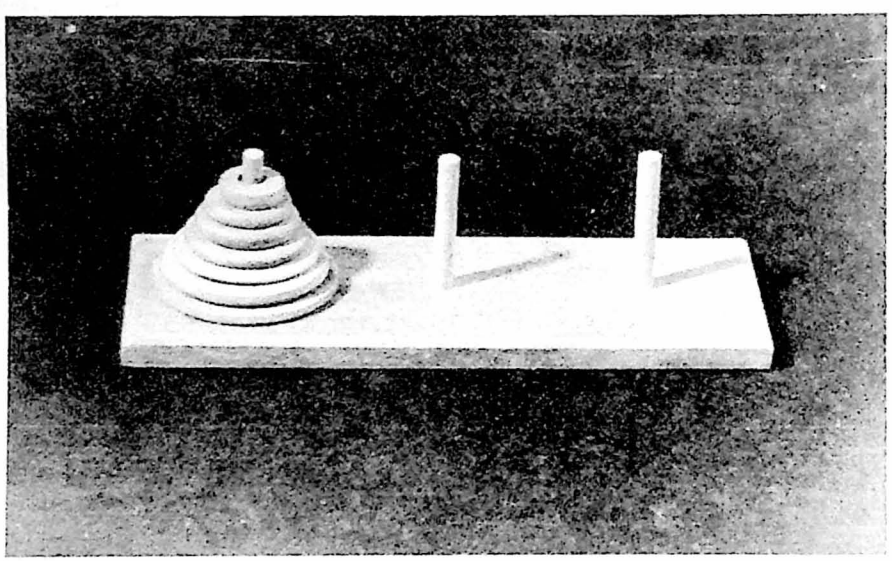
Wenn möglich, sollten Rekursionen vermieden werden, denn jede Rekursion läßt sich durch eine Iteration ersetzen.

Wenn jedoch die Rekursion der Iteration durch Klarheit in der Struktur und Einfachheit deutlich überlegen ist, verwenden wir die Rekursion.

Türme von Hanoi

11.3 Beispiele zu Rekursionen

1. Beispiel:



Ziel

Einigen ist sicherlich das Spiel „Türme von Hanoi" bekannt. Eine Holzausführung dieses Spieles ist auf dem Foto dargestellt. Ziel des Spieles: Bringe alle Scheiben von einem Turm auf einen anderen.

Türme von Hanoi

Spielregeln: Bewege jeweils nur eine Scheibe von einem Turm zu einem anderen. Lege stets eine kleinere auf eine größere Scheibe, nie umgekehrt.

Regeln

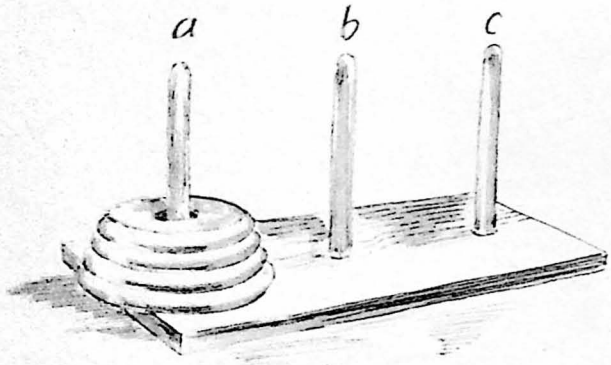
Das Spiel läßt sich im Prinzip mit beliebig vielen Scheiben spielen. Auf der Abbildung sind es sieben.

An dieser Stelle sollten wir den Computer mal abschalten und das Buch beiseite legen, um ein wenig zu spielen. Auch mit einigen unterschiedlich großen Münzen läßt sich „Türme von Hanoi" spielen, wenn keine andere Ausführung vorhanden ist.

Spielen

Aufgabe: Spielen Sie „Türme von Hanoi" mit 4 Scheiben. Versuchen Sie, so wenig Züge wie möglich zu machen. Schreiben Sie die Züge auf.

Wir nennen die Türme a, b und c.



Nun wollen wir uns ein Programm schreiben, das uns die Züge für dieses Spiel mit vorgegebener Scheibenzahl ausgibt.

Programmplanung

Beim realen Spiel wird man sicherlich erst einmal mit wenigen Scheiben anfangen. So leicht wollen wir es uns nicht machen.

Nehmen wir an, unser Spiel hat 70 Scheiben. Diese 70 Scheiben sollen von Turm a nach Turm b gebracht werden. Dies kann nicht auf einmal geschehen, da nur jeweils eine Scheibe bewegt werden darf.

→ Der Lösungsalgorithmus, der hier verfolgt werden soll, ist folgender: Wir bringen erst einmal alle Scheiben, mit Ausnahme der untersten, auf den Hilfsturm. Dann bewegen wir die unterste Scheibe auf den Zielturm. Schließlich bringen wir die Scheiben vom Hilfsturm auf den Zielturm.

Lösungsalgorithmus

In unserem Beispiel heißt das: Bringe 69 Scheiben auf Turm c. Bewege eine Scheibe auf Turm b. Bringe 69 Scheiben auf Turm b. Hinter dem „Bringen von 69 Scheiben" verbirgt sich sicherlich mehr als nur ein Befehl, nämlich ein sehr ähnlicher Algorithmus mit 68 Scheiben.

Wir planen eine Prozedur: Bringe (70, a, b, c).
Diese Prozedur soll folgendes ausführen: Bringe 70 Scheiben von Turm a nach Turm b unter Zuhilfenahme von Turm c als Zwischenlager für die 69 Scheiben.

Planung

Türme von Hanoi

Folgende Anweisungen führt die Prozedur „Bringe" aus:

Prozedur

```
Bringe (70, a, b, c) : Bringe (69, a, c, b)
                       Bewege eine Scheibe von a nach b
                       Bringe (69, c, b, a)
```

In Worten: Bringe 70 Scheiben von a nach b über c heißt: Bringe 69 Scheiben von a auf den Hilfsturm c unter Zuhilfenahme von b. Bewege eine Scheibe (die unterste) von a nach b. Bringe die 69 Scheiben vom Hilfsturm c nach b unter Zuhilfenahme von a.

Rekursion

Jetzt hat es jeder gemerkt: Wir benutzen eine Rekursion. Die Prozedur „Bringe" ruft sich selber zweimal auf.

Schauen wir uns noch an, was dann Bringe (69, a, c, b) macht:

```
Bringe (69, a, c, b) : Bringe (68, a, b, c)
                       Bewege eine Scheibe von a nach c
                       Bringe (68, b, c, a)
```

Aufgabe: Was macht Bringe (69, c, b, a)?

Allgemein können wir für n Scheiben schreiben:

```
Bringe (n, a, b, c) : Bringe (n−1, a, c, b)
                      Bewege eine Scheibe von a nach b
                      Bringe (n−1, c, b, a)
```

Abbruch-bedingung

Da wir uns jetzt mit Rekursionen auskennen, wissen wir, daß jede Rekursion eine Abbruchbedingung braucht.

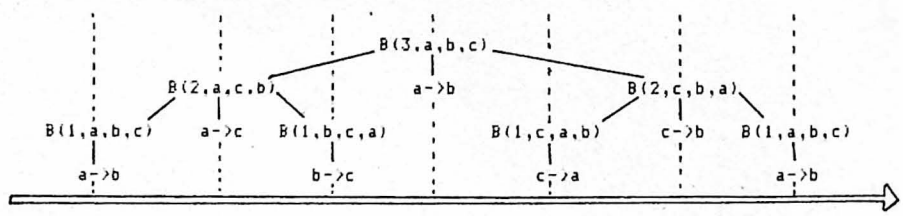
In unserem Fall ist es natürlich nur sinnvoll, weitere Prozeduren mit n−1 Scheiben aufzurufen, wenn die Anzahl der Scheiben größer als Null ist.

Analyse

Bevor wir uns nun an ein Pascal-Programm heranmachen, sollten wir an einem einfachen Beispiel die Prozedur „Bringe" testen.

Wir bringen 3 Scheiben von a nach b über c. Dazu heißt B(3,a,b,c) soviel wie Bringe(3,a,b,c) und a–>b soviel wie Bewege eine Scheibe von a nach b.

Züge



Die einzigen Operationen, die überhaupt ausgeführt werden können, sind die Bewegungen einer Scheibe (z.B. a−>b). Lesen wir diese Bewegungen in der Reihenfolge ihrer Ausführung (Hilfe: senkrechte Ebenen von links nach rechts), so erhalten wir:

Türme von Hanoi

a->b
a->c
b->c
a->b
c->a
c->b
a->b

Züge

Aufgabe: Schreiben Sie ein solches Diagramm für 4 Scheiben.

Nun aber an die Pascal-Lösung:

Prozedur

```
PROCEDURE Bringe (n : INTEGER; a, b, c : CHAR);

  BEGIN
    IF n > 0 THEN BEGIN
                    Bringe (n-1, a, c, b);
                    WRITELN (a, ' -> ', b);
                    Bringe (n-1, c, b, a)
                  END
  END; (* von Bringe *)
```

Das ist schon die ganze Prozedur (wie oben geplant). Die Variablen für die Türme sind vom Typ CHAR. Weisen wir ihnen im Programm der Einfachheit halber die Buchstaben a, b und c zu.

Programm

```
PROGRAM Hanoi;

  VAR Anzahl : INTEGER;

  PROCEDURE Bringe (n : INTEGER; a, b, c : CHAR);

    BEGIN
      IF n > 0 THEN BEGIN
                      Bringe (n-1, a, c, b);
                      WRITELN (a, ' -> ', b);
                      Bringe (n-1, c, b, a)
                    END
    END; (* von Bringe *)

  BEGIN (* Hauptprogramm *)
    WRITELN ( Tuerme von Hanoi );
    WRITE ('Eingabe der Scheibenzahl:  ');
    READLN (Anzahl);
    Bringe (Anzahl, 'a', 'b', 'c')
  END.
```

Aufgabe: Testen Sie das Programm mit 3 und 4 Scheiben, und vergleichen Sie mit den oben zu Fuß erstellten Lösungen.

Testen Sie nun mit mehr Scheiben, und spielen Sie nach.

(Tip: Da es zu sehr vielen Zügen kommen kann, sollte das Programm eventuell für einen Drucker umgeschrieben werden:

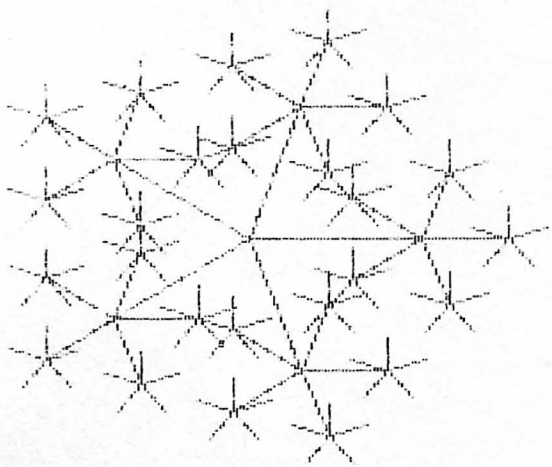
Programm für Drucker

```
PROGRAM Hanoi;

  VAR Anzahl : INTEGER;
      Drucker : INTERACTIVE;

  PROCEDURE Bringe (n : INTEGER; a, b, c : CHAR);
```

(Fortsetzung des Programms)

```
    BEGIN
      IF n > 0 THEN BEGIN
                      Bringe (n-1, a, c, b);
                      WRITELN (Drucker, a, ' -> ', b);
                      Bringe (n-1, c, b, a)
                    END
    END; (* von Bringe *)

BEGIN (* Hauptprogramm *)
  WRITELN ('Tuerme von Hanoi');
  WRITE ('Eingabe der Scheibenzahl: ');
  READLN (Anzahl);
  REWRITE (Drucker, 'PRINTER:');
  Bringe (Anzahl, 'a', 'b', 'c');
  CLOSE (Drucker)
END.
```

Schneeflocke *2. Beispiel:*



3, 140, 90, 50

Rekursion Diese herrliche Grafik läßt sich ebenfalls mit einer Rekursion erzeugen.

Die Grundfigur (Ordnung=1) ist ein strahlenförmiges Gebilde, dessen 5 Seiten sich jeweils um einen Winkel von 72 Grad unterscheiden. An jede der Seiten wird (bei höheren Ordnungen) wieder ein solches Gebilde, allerdings mit halber Seitenlänge, gezeichnet.

Programm

```
PROGRAM Schneeflocke; (* Idee: F.Binner, J.Moldenhauer *)

  USES TURTLEGRAPHICS;
  VAR Ordnung, X, Y, Grundseite : INTEGER;

  PROCEDURE Flocke (Grad, x, y, L : INTEGER);  (* rekursiv *)
    VAR i : INTEGER;
    BEGIN
      IF Grad=1 THEN BEGIN
        MOVETO (x,y);
        TURNTO (90);
```

(Fortsetzung des Programms)

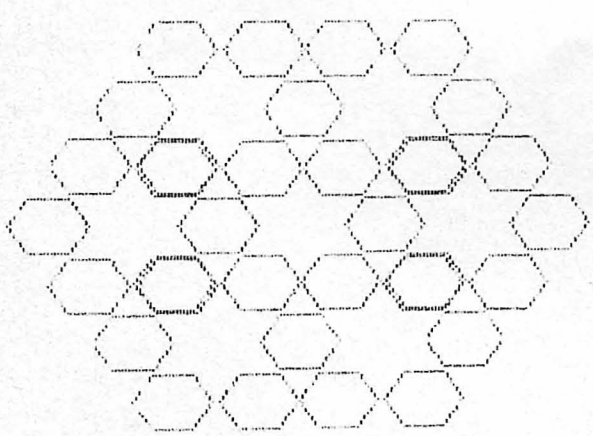
```
    FOR i:=1 TO 5 DO BEGIN
                       MOVE (L);
                       MOVETO (x,y);
                       TURN (72)
                     END (* von FOR *)
  END (* von IF *)
  ELSE BEGIN
    TURNTO (0);
    FOR i:=1 TO 5 DO BEGIN
                       MOVE (L);
                       Flocke (Grad-1, TURTLEX, TURTLEY, L DIV 2);
                       MOVETO (x,y);
                       TURNTO (i * 72)
                     END (* von For *)
  END (* von ELSE *)
END; (* von Flocke *)

BEGIN (* Hauptprogramm *)
  WRITELN ('Schneeflocken: ');
  WRITELN ('Geben Sie die Ordnung (Rekursionstiefe),');
  WRITELN ('die Anfangskoordinaten und Anfangslaenge ein.');
  WRITE ('Ordnung: ');
  READLN (Ordnung);
  WRITE ('       X: ');
  READLN (X);
  WRITE ('       Y: ');
  READLN (Y);
  WRITE (' Laenge: ');
  READLN (Grundseite);
  INITTURTLE;
  PENCOLOR (NONE);
  MOVETO (X,Y);
  PENCOLOR (WHITE);
  Flocke (Ordnung, X, Y, Grundseite);
  READLN (* zum Abwarten *)
END.
```

Aufgabe: Spielen Sie mit dem Programm!

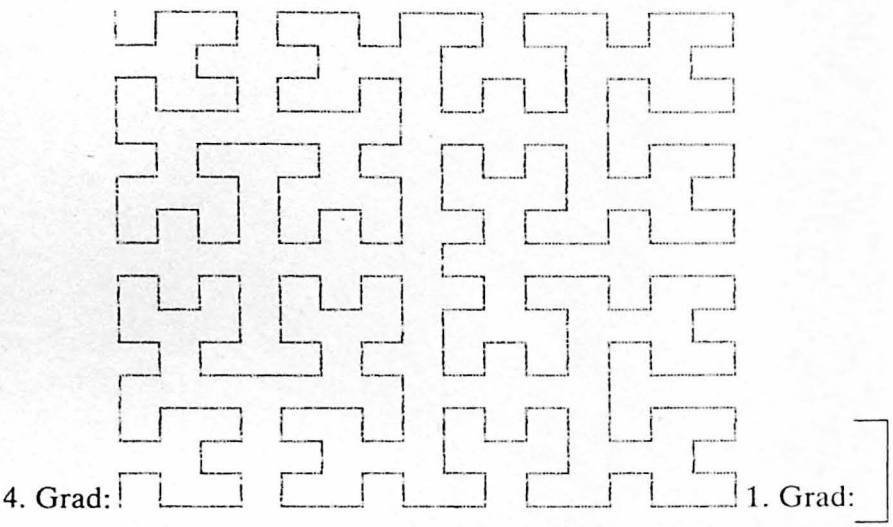
Übungen:

11.3.1 Schreiben Sie ein rekursives Programm, das (ähnlich dem Programm „Schneeflocke") folgende Grafik erstellt.

(Hier 3. Grad.)



11.3.2 Entwerfen Sie weitere rekursive Grafiken (z. B. „Hilbert", Lösung siehe APPLE3:).



11.4 Zusammenfassung

Mit den Rekursionen haben wir ein algorithmisch sehr elegantes und interessantes Mittel kennengelernt, entsprechend geeignete Probleme zu lösen. Allerdings haben wir auch feststellen müssen, daß es einige Probleme gibt, in denen man Rekursionen besser vermeidet und statt dessen Iterationen benutzt.

Eine Programmiersprache wie Pascal macht es uns leicht, Rekursionen zu handhaben. Diese Möglichkeit ist nicht in allen Sprachen selbstverständlich. So muß man z.B. in den meisten BASIC-Versionen unter ziemlichen Einschränkungen arbeiten (z.B. muß man die Stackverwaltung selbst simulieren, und es gibt nur globale Variablen).

Eindimensionale Felder

Kapitel 12

12. Eindimensionale Felder

Dieses Kapitel ist ein kurzgefaßter Vorgriff auf die Behandlung von Feldern allgemein. Für sehr viele Probleme und Algorithmen ist es vorteilhaft, wenn nicht gar unerläßlich, mit sogenannten indizierten Variablen zu arbeiten. Daher werden hier die eindimensionalen Felder ohne viel theoretischen Hintergrund eingeführt und später in Zusammenhang mit den mehrdimensionalen Feldern noch einmal aufgegriffen (Band 2).

Nach dem Lesen dieses Kapitels

- wissen wir, was unter indizierten Variablen zu verstehen ist.
- kennen wir die Definition eines eindimensionalen Feldes im Deklarationsteil.
- können wir mit einfachen eindimensionalen Feldern umgehen.

Schülerliste

1. ***Spinni Winni***
2. ***Fritz der Clown***
3. ***Fix***
4. ***Foxy***
5. ***Donald Duck***
6. :
   :
   :
   :
   :

12.1 Wenn mehrere Variablen den gleichen Namen haben sollen...

Aufgabe (Schülerliste)

Wir wollen eine Namensliste aller Schüler unserer Klasse erstellen. Nehmen wir an, es wären maximal 40 Schüler in einer Klasse zulässig (die Pädagogen mögen es mir verzeihen!). In unserer Klasse sind aber nur 21 Schüler.

Wir könnten als Variablen für die Schüler die Namen Schueler1, Schueler2, ..., Schueler40 erfinden.

Wie aber sollen wir nun alle Schülernamen eingeben und später wieder ausgeben? Diese Methode zwingt uns zu 40 Ein- und Ausgabeanweisungen. Wahnsinn!

Natürlich geht es in Pascal (wie in allen Programmiersprachen) einfacher. Wir können indizierte Variablen verwenden. Dabei heißen in unserem Beispiel alle Variablen Schueler, gefolgt von einer Nummer 1 bis 40.

Deklaration

Im Deklarationsteil des Programms müssen wir diesen Datentyp als Feld (Array) erklären.

```
VAR Schueler : ARRAY [1..40] OF STRING;
```

Die Variable Schueler kann also die Nummern 1 bis 40 haben, wobei jede dieser Variablen vom Typ STRING ist.

In den Anweisungen werden die Variablen als Schueler[1], Schueler[2], ... , Schueler[40] verwendet.

Die Zahl in der Klammer wird Index genannt. Als Index kann auch eine Variable vom Datentyp INTEGER verwendet werden.

Beispiel:

Zuweisung

```
I := 5;
READLN (Schueler[I]);
```

hat den gleichen Effekt wie:

```
READLN (Schueler[5]);
```

Hinweis: In Programmiersprachen, die mit einem Compiler übersetzt werden, muß bei der Deklaration eines Feldes *genau* festgelegt werden, welche Nummern als Indizes auftreten können, damit der Compiler genau so viele Speicherplätze für die indizierten Variablen bereitstellen kann.

Planen wir unser Programm für eine Schülerliste:

Blockdiagramm

Programm LISTE	
Konstante	N=40
Variablen	i, e : ganze Zahlen Schueler : Feld von 1 bis N von Zeichenketten

(Fortsetzung des Blockdiagramms)

Anfang
Eingabe der Schuelerzahl e
Zähle i von 1 bis e und tue
Eingabe Schueler[i]
Zähle i von 1 bis e und tue
Ausgabe Schueler[i]
Ende.

Das Programm:

Programm

```
PROGRAM Liste;

  CONST N = 40;
  VAR i, e : INTEGER;
      Schueler : ARRAY [1..N] OF STRING;

  BEGIN
    WRITE ('Schuelerzahl: ');
    READLN (e);
    WRITELN ('Eingabe:');
    FOR i:=1 TO e DO BEGIN
      WRITE ('Schueler Nr. ',i:2,': ');
      READLN (Schueler[i])
    END; (* von FOR i *)
    WRITELN ('Ausgabe:');
    FOR i:=1 TO e DO WRITELN (Schueler[i])
  END.
```

Aufgabe: Schreiben Sie das Programm so um, daß nur die Schüler mit einem vorher einzugebenden Anfangsbuchstaben ausgegeben werden.

Beispiele für richtige Felddeklarationen:

Richtige Beispiele

```
CONST Anfang=7;
      Ende=15;

VAR A : ARRAY [1..30] OF INTEGER;
    B : ARRAY [2..Anfang] OF REAL;
    C : ARRAY [Anfang .. Ende] OF CHAR;
    D : ARRAY [20..25] OF BOOLEAN;
```

Eine falsche Deklaration ist:

Falsches Beispiel

```
VAR N : INTEGER;
    E : ARRAY [1..N] OF REAL;
falsch, weil N keine Konstante ist.
```

Definition ARRAY

Eindimensionales Feld:

Eine Variable F, die als

F : ARRAY [A .. E] OF Datentyp;

deklariert ist, stellt ein eindimensionales Feld dar. A und E sind Konstanten vom Datentyp INTEGER, BOOLEAN, CHAR oder Aufzählungstyp (Band 2). Sie geben den Anfangs- (A) und Endindex (E) der Variablen F an.

Jede einzelne Variable F [x] (x ist vom Typ der Konstanten, und es gilt: A <= i <= E) ist vom oben deklarierten Datentyp. Der Datentyp sollte an dieser Stelle ein einfacher Datentyp (Kap. 5) sein.

Übungen:

12.1.1 Was ist der Unterschied zwischen folgenden Konstruktionen:

Zahl(5) und Zahl[5]?

12.1.2 Warum kann man die Größe des Feldes nicht variabel deklarieren?

12.1.3 Ist folgende Felddeklaration erlaubt?

```
VAR Alpha : ARRAY ['a'..'z'] OF CHAR;
```

12.2 Beispiele eindimensionaler ARRAYs

1. Zufallstest mit Histogramm

Aufgabe (Zufallstest)

Wir wollen die Zuverlässigkeit unseres Zufallszahlengenerators testen. Dazu erzeugen wir Zufallszahlen zwischen 0 und 9 und stellen deren Häufigkeit in einem sogenannten Histogramm (Balkendiagramm) dar. Mit einem 80-Zeichenbildschirm könnten wir bis zu 75 Kreuze in der Breite auf dem Bildschirm ausgeben, wobei jedes Kreuz für das Vorkommen der entsprechenden Zufallszahl steht. Einige Zeichen müssen wir für die Ziffern freilassen.

Histogramm

Programm

```
PROGRAM Randomtest;

  USES APPLESTUFF; (* fuer die Zufallszahlen *)

  CONST Breite = 75;  (* bei anderen Bildschirmen aendern *)
  VAR Maximum, i, j, n : INTEGER;
      Z : ARRAY [0..9] OF INTEGER;

  BEGIN
    PAGE (OUTPUT);
    RANDOMIZE;  (* neue Zufallszahlen *)
    WRITELN; WRITELN;
    Maximum := 0;
    FOR i:=0 TO 9 DO Z[i] := 0;  (* alle Zahlen auf 0 setzen *)
    REPEAT
      n := RANDOM MOD 10; (* Zahl zwischen 0 und 9 *)
      Z[n] := Z[n] + 1;  (* Anzahl erhoehen *)
      IF Z[n] > Maximum THEN Maximum := Z[n]  (* Maximum der Anzahlen
                                          auf neuesten Stand bringen *)
    UNTIL Maximum >= Breite;  (* mehr Kreuze passen nicht auf Bild *)
    FOR i:=0 TO 9 DO BEGIN
      WRITE (i:1, ':');
      FOR j:=1 TO Z[i] DO WRITE ('X');  (* Anzahl der Kreuze fuer i *)
      WRITELN; WRITELN
    END (* von FOR i *)
  END.
```

Das Programm könnte folgendes Ergebnis haben:

Ergebnis

```
0:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
1:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
3:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
4:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
5:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
6:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
7:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
9:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

2. Lottozahlen

Aufgabe (Lottozahlen)

Das folgende Programm hilft uns beim Ausfüllen eines Lottoscheins (natürlich ohne Gewähr).

Dazu werden sechs Zahlen L[1] bis L[6] als Zufallszahlen erzeugt, wobei keine Mehrfachnennungen vorkommen dürfen.

Anschließend wird der Lottoschein „ausgefüllt":

	10	20	30	40
1	11	21	X	41
2	12	22	32	42
X	13	23	33	43
4	14	24	34	44
5	X	25	35	45
6	16	26	X	46
7	17	27	37	X
8	18	28	38	48
9	19	X	39	49

Lottozahlen

Das Programm dazu:

Programm

```
PROGRAM Lottoblock;

  USES APPLESTUFF; (* fuer Zufallszahlen *)

  VAR i, j, z : INTEGER;
      L : ARRAY [1..6] OF INTEGER;
      Kreuz, Neuezahl, Ende : BOOLEAN;

  BEGIN
    RANDOMIZE; (* neue Zufallszahlen *)
    FOR i:=1 TO 6 DO BEGIN
      REPEAT
        z := RANDOM MOD 49 + 1;  (* Zahl zwischen 1 und 49 *)
        Neuezahl := TRUE;
        FOR j:=1 TO i DO IF z = L[j] THEN Neuezahl := FALSE
          (* ist wahr, wenn z neue Zufallszahl ist *)
      UNTIL Neuezahl;
      L[i] := z
    END; (* von FOR i *)
    PAGE (OUTPUT);
    WRITELN ('Die Lottozahlen:');
    WRITELN;
    WRITE ('   ');
    Ende := FALSE;
    i := 10;
    REPEAT
      Kreuz := FALSE;
      FOR j:=1 TO 6 DO IF L[j]=i THEN Kreuz := TRUE;
      IF Kreuz THEN WRITE ('  X')
               ELSE WRITE (i:3);
      IF i >= 49 THEN Ende := TRUE;
      IF i >= 40 THEN BEGIN
                        WRITELN;
                        i := i-39
                      END
                 ELSE i := i+10
    UNTIL Ende;
    for i:=1 to 6 do write (L[i]:4);
  END.
```

3. Größter und kleinster Meßwert

Aufgabe (Meßwertanalyse)

In dem folgenden Beispiel soll eine Reihe von Meßwerten (Datentyp REAL) eingegeben und der größte und kleinste Meßwert bestimmt werden. Außerdem berechnen wir als Nebenprodukt den Mittelwert der Meßwerte und die größte Abweichung vom Mittelwert.

Programm

Das Programm:

```
PROGRAM Messwerte;

  CONST N = 50;
  VAR Max, Min, Mittelwert, Abweichung, Dif : REAL;
      i, Anz : INTEGER;
      Wert : ARRAY [1..N] OF REAL;

  BEGIN
    WRITELN ('Messwerterfassung und Analyse: ');
    WRITE ('Wieviele Me"werte (max. 50) : ');
    READLN (Anz);
    FOR i:=1 TO Anz DO BEGIN
      WRITE (i:2, '.Wert: ');
      READLN (Wert[i])
    END; (* von FOR i *)
```

(Fortsetzung des Programms)

```
    Max := Wert[1];
    Min := Max;       (* bestimmte Wert zuweisen *)
    Mittelwert := 0;
    FOR i:=1 TO Anz DO BEGIN
      IF Wert[i] > Max THEN Max := Wert[i];
      IF Wert[i] < Min THEN Min := Wert[i];
      Mittelwert := Mittelwert + Wert[i]   (* vorlaeufig *)
    END; (* von FOR i *)
    Mittelwert := Mittelwert / Anz;
    Abweichung := 0;
    FOR i:=1 TO Anz DO BEGIN
      Dif := ABS (Mittelwert - Wert[i]);
      IF Dif > Abweichung THEN Abweichung := Dif
    END; (* von FOR i *)
    PAGE (OUTPUT);
    WRITELN; WRITELN;
    WRITELN ('Das Maximum ist: ', Max:7:2);
    WRITELN ('Das Minimum ist: ', Min:7:2);
    WRITELN ('Der Mittelwert : ', Mittelwert:7:2);
    WRITELN ('Mit der groessten Abweichung von: ');
    WRITELN ('+/-', Abweichung:7:2)
  END.
```

Übungen:

12.2.1 Schreiben Sie ein Programm „Notenspiegel" mit folgender Aufgabe:

Eingabe der Anzahlen für die Noten 1 bis 6.

Erstellen eines Histogramms.

Berechnung der Durchschnittsnote.

12.2.2 Ein Feld mit N Elementen (INTEGER) wird eingegeben. Das Programm soll feststellen, ob das Feld sortiert ist.

12.2.3 Schreiben Sie ein Programm zur Umwandlung von Dezimal- in Hexadezimalzahlen und umgekehrt.

(Hexadezimalzahlen sind Zahlen zur Basis 16 mit folgenden Ziffern: 0 1 2 3 4 5 6 7 8 9 A B C D E F. Sie werden häufig zur Programmierung in Maschinensprache verwendet.)

12.2.4 Durch einen Abzählreim von A Worten Länge wird die Reihenfolge für ein Spiel bestimmt. Dazu werden die Namen der N Kinder zuerst nacheinander eingegeben. Den Kindern werden die Eigenschaften 'Drin=True' oder 'Drin=FALSE' zugewiesen. Dabei soll 'Drin' ein ARRAY vom Typ BOOLEAN sein mit N Elementen.

Beim Durchzählen der Worte 1 bis A (Achtung: Wenn A überschritten, dann bei 1 anfangen) fällt jedesmal ein Kind heraus und wird ausgegeben. So entsteht eine neue Liste in der Abzählreihenfolge.

12.3 Zusammenfassung

Das vorliegende Kapitel über eindimensionale Arrays ist ein eingeschränkter Vorgriff auf mehrdimensionale Arrays. Die eindimensionalen Felder sind für viele Probleme aus der Algorithmik sehr nützlich und unentbehrlich. So brauchen wir sie für jede Art von Listen und zum Sortieren von Datensätzen.

Der Datentyp Array ist im Grunde einfach zu verstehen und sollte in vielfältigen Programmen angewendet werden, um zu einem sicheren Umgang damit zu gelangen.

Wer Ordnung hält, ist nur zu faul zum Suchen – Sortieralgorithmen

Kapitel 13

In diesem Kapitel wollen wir uns einem Standardthema der Informatik, dem Sortieren von Daten zuwenden.

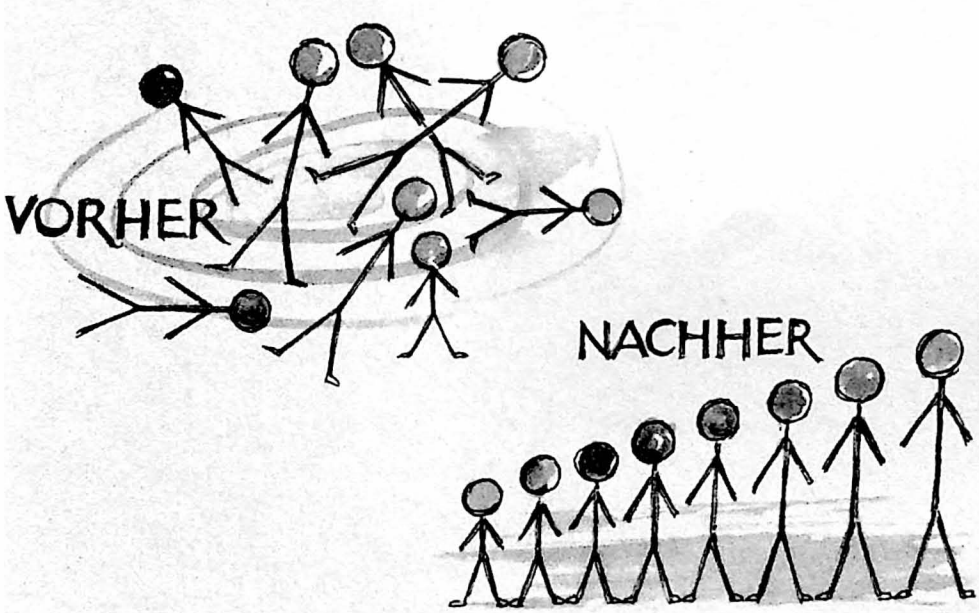
Dafür brauchen wir indizierte Variablen (also eindimensionale Felder), damit wir jeder Variablen eine Nummer zuweisen können. Dann werden die Daten in aufsteigender Reihenfolge der Größe nach sortiert. Außerdem werden Schleifen (insbesondere die FOR-Schleife) benötigt. Wir werden drei verschiedene Sortieralgorithmen kennenlernen und dabei feststellen, daß sich ein und dasselbe Problem mit mehreren Algorithmen lösen läßt.

Nebenbei: Es gibt noch mehr Lösungen. Dazu sollte man aber die einschlägige Fachliteratur lesen (siehe Literaturliste, bes. N. Wirth, Algorithmen und Datenstrukturen).

Der dritte Algorithmus (Quicksort) setzt schon das Wissen über Rekursionen voraus. Sollten Rekursionen überschlagen sein, so muß dieser Algorithmus vorerst ausgelassen werden. Allerdings kann man einen Algorithmus auch ohne Verständnis einfach benutzen.

Nach dem Lesen dieses Kapitels

- kennen wir mehrere Lösungen für das Problem des Sortierens.
- Wissen wir, daß verschiedene Algorithmen für ein und dieselbe Problemstellung unterschiedliche Qualitäten haben.



13.1 Sortieren durch Austausch (1)

Beispielzahlenfolge

55	94	67	44	12	6	18	42

Aufgabe (Sortieren)

Diese acht Zahlen wollen wir sortieren.

Wir würden möglicherweise so vorgehen, daß wir uns erst die kleinste Zahl suchen, sie aufschreiben, dann die kleinste der verbleibenden Zahlen usw.

Natürlich überblicken wir sofort, welche die kleinste Zahl ist. Dies ist aber nur daher möglich, weil es so wenige Zahlen sind. Hätten wir 10 000 Zahlen zu ordnen, so müßten wir uns schon eine andere Lösung suchen.

Wir geben unseren Zahlen Namen mit Nummern:

a1	a2	a3	a4	a5	a6	a7	a8
55	94	67	44	12	6	18	42

Lösungsidee

Nun behaupten wir, die Zahl mit der kleinsten Nummer, a1, sei die kleinste Zahl. Natürlich sieht jeder, daß das hier nicht zutrifft.

Tauschen per Hand

Daher müssen wir die Zahl a1 mit allen folgenden (a2,a3,a4,a5,a6, a7,a8) vergleichen und den Platz tauschen, wenn wir eine kleinere Zahl als a1 gefunden haben:

a1 vergleichen mit a2, OK
a1 vergleichen mit a3, OK
a1 vergleichen mit a4, vertauschen a1 ↔ a4

a1	a2	a3	a4	a5	a6	a7	a8
44	94	67	55	12	6	18	42

a1 vergleichen mit a5, vertauschen a1 ↔ a5

a1	a2	a3	a4	a5	a6	a7	a8
12	94	67	55	44	6	18	42

a1 vergleichen mit a6, vertauschen a1 ↔ a6

a1	a2	a3	a4	a5	a6	a7	a8
6	94	67	55	44	12	18	42

a1 vergleichen mit a7, OK
a1 vergleichen mit a8, OK

Kleinste Zahl gefunden

Damit haben wir die kleinste Zahl a1=6 gefunden.

Bei den Vertauschungen, die nötig waren, fällt übrigens gleich der entscheidende Mangel dieses Algorithmus auf: Die zweitkleinste Zahl 12, die schon einmal an vorderster Stelle stand, ist durch die Vertauschungen wieder weit nach hinten gerutscht, ähnlich wie andere kleinere Zahlen.

Algorithmus

Nun müssen wir die zweitkleinste Zahl a2 finden:

a2	a3	a4	a5	a6	a7	a8
94	67	55	44	12	18	42

a2 vergleichen mit a3, vertauschen a2 ↔ a3

a2	a3	a4	a5	a6	a7	a8
67	94	55	44	12	18	42

a2 vergleichen mit a4, vertauschen a2 ↔ a4

a2	a3	a4	a5	a6	a7	a8
55	94	67	44	12	18	42

a2 vergleichen mit a5, vertauschen a2 ↔ a5

a2	a3	a4	a5	a6	a7	a8
44	94	67	55	12	18	42

a2 vergleichen mit a6, vertauschen a2 ↔ a6

a2	a3	a4	a5	a6	a7	a8
12	94	67	55	44	18	42

a2 vergleichen mit a7, OK.
a2 vergleichen mit a8, OK.

Damit ist a2=12 die zweitkleinste Zahl.

Zweitkleinste Zahl gefunden

Nun wird a3 als die drittkleinste Zahl gesucht. Dazu vergleichen wir a3 mit allen folgenden a4 bis a8 und vertauschen, wenn nötig.

Wir finden a3=18 als dritte Zahl.

Es bleibt:

a4	a5	a6	a7	a8
94	67	55	44	42

Die Prozedur des Vergleichens und Vertauschens wird fortgeführt, bis wir a8 vergleichen müßten. Dies ist selbstverständlich nicht mehr nötig, da a8 nur noch die größte Zahl sein kann.

Aufgabe: Schreiben Sie die Vergleiche und nötigen Vertauschungen für a3 bis a8 ebenfalls auf.

Was haben wir nun getan?

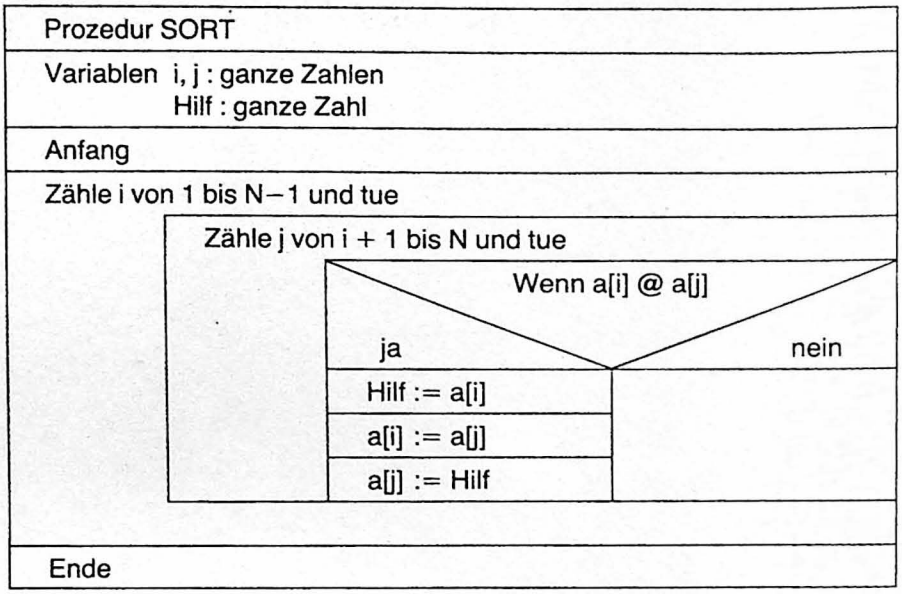
Algorithmus

Wir haben nacheinander die Zahlen a1 bis a7 mit allen jeweils folgenden Zahlen verglichen. Wenn dabei eine kleinere Zahl gefunden wurde, haben wir die Zahlen vertauscht.

Dazu planen wir eine Prozedur namens Sort in Pascal. Die Variablen im Hauptprogramm sollen sein a[1], a[2], ... , a[N], wobei N eine Konstante mit dem Wert 8 ist.

Tauschsort 1

Blockdiagramm



Programm Das Programm mit der Prozedur Sort:

```
PROGRAM Tauschsort1;

  CONST N = 8;
  VAR  i : INTEGER;
       a : ARRAY [1..N] OF INTEGER;

  PROCEDURE Sort;
    VAR i, j : INTEGER;
        Hilf : INTEGER;

    BEGIN
      FOR i:=1 TO N-1 DO
        FOR j:=i TO N DO
          IF a[i] > a[j] THEN BEGIN
            Hilf := a[i];
            a[i] := a[j];
            a[j] := Hilf      (* Speichertausch *)
          END (* von IF *)
    END; (* von Sort *)

  BEGIN (* Hauptprogramm *)
    WRITELN ('Geben Sie ', N:3, ' Zahlen ein: ');
    FOR i:=1 TO N DO READLN (a[i]);
    Sort;
    WRITELN ('Sortiert: ');
    FOR i:=1 TO N DO WRITELN (a[i]:3)
  END.
```

In der Sortierprozedur werden also die Zahlen a[1] bis a[7] mit allen folgenden Zahlen a[i+1] bis a[8] verglichen und nötigenfalls vertauscht.

Analyse Damit wir die Vorgänge bei den Vergleichen und Vertauschungen besser verstehen können, hier ein etwas erweitertes Programm, das die Zahlen bei jedem Schleifendurchlauf auf einen Drucker ausdruckt. Mit dem Programm könnte auch eine Teilfolge sortiert werden, da der Anfangsindex l (für links) und der Endindex r (für rechts) gesetzt wird. Im Ausdruck wird jeweils der Wert für l, r, i und j ausgegeben. In der Zahlenfolge werden die Zahlen unterstrichen,

Tauschsort 1 mit Druckausgabe

die vertauscht worden sind. Zählt man nach, so ergibt sich bei unserer Beispielzahlenfolge eine Anzahl von 21 Vertauschungen. Das bedeutet, daß 21mal Daten transportiert werden müssen.

Programm (für Drucker) Ausdruck

Hinweis zum Programm: Bei dem verwendeten Drucker wird das Unterstreichen mit der Kontrollzeichensequenz: ESC, '–', CHR(0/1) gesteuert. Für andere Drucker müssen diese Programmzeilen geändert werden.

Programm mit Druckerausgabe

```
PROGRAM Tauschsort1;

  CONST N=8;
  VAR i : INTEGER;
      a : ARRAY [1..N] OF INTEGER;
      d : INTERACTIVE;  (* Drucker *)

  PROCEDURE Ausgabe (l,r,i,j : INTEGER);
    VAR k : INTEGER;
    BEGIN
      WRITE (d,CHR(27),'-',CHR(1));  (* Unterstreichen an *)
      WRITE (d,'  l:  r:  i:  j:          ');
      FOR k:=1 TO N DO WRITE (d,k:4);
      WRITE (d,CHR(27),'-',CHR(0));  (* Unterstreichen aus *)
      WRITELN (d);
      WRITE (d,l:4, r:4, i:4, j:4);
      WRITE (d,'    a[i]:  ');
      FOR k:=1 TO N DO BEGIN
        IF (k=i) OR (k=j) THEN BEGIN
          WRITE (d,CHR(27),'-',CHR(1)); (* Unterstreichen an *)
          WRITE (d,a[k]:4);
          WRITE (d,CHR(27),'-',CHR(0))  (* Unterstreichen aus *)
        END (* von IF *)
        ELSE WRITE (d,a[k]:4)
      END; (* von FOR k *)
      WRITELN (d);
      WRITELN(d)
    END; (* von Ausgabe *)

  PROCEDURE Sort (l, r : INTEGER);
    VAR i, j : INTEGER;
        w : INTEGER;

    BEGIN
      FOR i:=l TO r-1 DO
        FOR j:=i TO r DO BEGIN
          IF a[i] > a[j] THEN BEGIN
            w := a[i]; a[i] := a[j]; a[j] := w;
            Ausgabe(l,r,i,j)
          END (* von IF *)
        END (* von FOR j *)
    END; (* von Sort *)

  BEGIN (* Hauptprogramm *)
    REWRITE (d, 'PRINTER:');
    WRITELN ('Geben Sie ',N:3, ' Zahlen ein.');
    FOR i:=1 TO N DO READLN (a[i]);
    WRITELN (d,'Anfang:');
    WRITE (d,'                              ');
    FOR i:=1 TO N DO WRITE (d,a[i]:4);
    WRITELN(d);
    Sort (1,N);
    WRITELN (d,'Ende:');
    WRITE (d,'                              ');
    FOR i:=1 TO N DO WRITE (d,a[i]:4);
    WRITELN(d);
    CLOSE (d)
  END.
```

Aufgabe: Testen Sie das Programm, und versuchen Sie anhand des Ausdrucks den Sortiervorgang zu verstehen!

Ausdruck

Andere Datentypen sortieren

Wenn wir eine andere Anzahl von Daten sortieren wollen, so ändern wir einfach die Konstante N.

Sollen Daten von anderen Typen sortiert werden, so wird der Datentyp des ARRAY *und* der Typ der Hilfsvariablen Hilf geändert.

Wenn wir Daten vom Typ CHAR oder STRING sortieren, dann wird die Reihenfolge des entsprechenden Zeichensatzes, hier ASCII (siehe Kap. 5.3), verwendet.

Übungen:

13.1.1 Schreiben Sie das Programm „Tauschsort 1" zum Sortieren von 10 Zeichen (Typ CHAR) um.

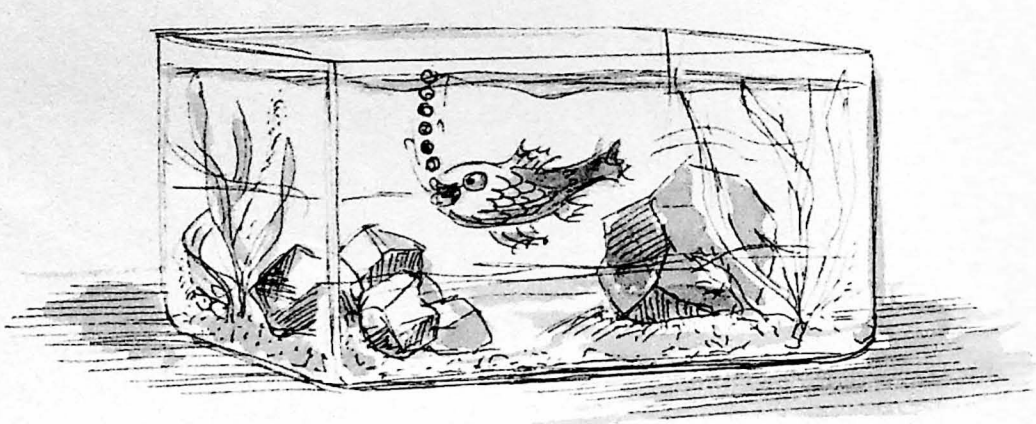
13.1.2 Schreiben Sie ein Programm zum Sortieren von 80 Ziffern, die zufällig erzeugt werden. Die Vertauschungen sollen auf dem Bildschirm sichtbar gemacht werden.

13.1.3 Schreiben Sie ein Programm, das 100 zufällig erzeugte Zeichen vom Typ CHAR sortiert. Ermitteln Sie die mittlere Zeit für 20 Programmläufe.

13.1.4 Schreiben Sie ein Programm zum Sortieren von höchstens 50 Namen. Dabei soll der Benutzer nach jeder Eingabe entscheiden, ob er noch einen Namen eingeben will.

13.1.5 Ändern Sie das Programm aus 13.1.3 dahingehend, daß bei jedem Suchlauf zugleich das größte *und* das kleinste Element gefunden wird. Wird das Programm dadurch schneller?

13.2 Sortieren durch Austausch (2) – Bubblesort



Im vorhergehenden Paragraphen haben wir einen einfach zu verstehenden Sortieralgorithmus kennengelernt, der einen entscheidenden Nachteil hat: Er bringt beim Vertauschen Daten, die schon an günstigen Positionen gestanden haben, an sehr ungünstige Positionen.

Tauschsort 2

Dadurch erhöht sich die Zahl der Vertauschungen unnötig. Aber gerade die mittlere Zahl von Vertauschungen ist ein wichtiges Qualitätsmerkmal für einen Sortieralgorithmus. Später wird es nämlich oft der Fall sein, daß hinter einem Element des Feldes, das sortiert werden soll, eine ganze Reihe von Daten steht. Dann braucht jede Vertauschung ungeheuer viel Zeit. Bei großen Datenmengen sollte also die Zahl der Vertauschungen gering gehalten werden.

Algorithmus

Unser Algorithmus in der Prozedur Sort kann leicht dadurch entscheidend verbessert werden, daß die innere Schleife, die die jeweils nachfolgenden Elemente untersucht, nicht vorwärts, sondern rückwärts durchlaufen wird.

Programm

Das Programm mit dieser Änderung:

```
PROGRAM Tauschsort2;

  CONST N = 8;
  VAR  i : INTEGER;
       a : ARRAY [1..N] OF INTEGER;

  PROCEDURE Sort;
    VAR i, j : INTEGER;
        Hilf : INTEGER;

    BEGIN
      FOR i:=1 TO N-1 DO
        FOR j:=N DOWNTO i DO          (* hier geaendert *)
          IF a[i] > a[j] THEN BEGIN
            Hilf := a[i];
            a[i] := a[j];
            a[j] := Hilf      (* Speichertausch *)
          END (* von IF *)
    END; (* von Sort *)

  BEGIN (* Hauptprogramm *)
    WRITELN ('Geben Sie ', N:3, ' Zahlen ein: ');
    FOR i:=1 TO N DO READLN (a[i]);
    Sort;
    WRITELN ('Sortiert: ');
    FOR i:=1 TO N DO WRITELN (a[i]:3)
  END.
```

Bubblesort

Der hier benutzte Sortieralgorithmus wird oft auch BUBBLESORT genannt. Beim Sortieren steigen die kleineren Zahlen wie Blasen (Bubbles) zum Anfang des Datenfeldes auf.

Analyse

Diese Eigenschaft des Sortieralgorithmus können wir uns auch hier wieder durch ein Demonstrationsprogramm auf einem Drucker sichtbar machen:

Programm (für Drucker)

```
PROGRAM Tauschsort2;

  CONST N=8;
  VAR i : INTEGER;
      a : ARRAY [1..N] OF INTEGER;
      d : INTERACTIVE;  (* Drucker *)

  PROCEDURE Ausgabe (l,r,i,j : INTEGER);
    VAR k : INTEGER;
    BEGIN
      WRITE (d,CHR(27),'-',CHR(1));      (* Unterstreichen an *)
      WRITE (d,'   l:  r:  i:  j:          ');
```

(Fortsetzung des Programms)

```
      FOR k:=1 TO N DO WRITE (d,k:4);
      WRITE (d,CHR(27),'-',CHR(0));      (* Unterstreichen aus *)
      WRITELN (d);
      WRITE (d,l:4, r:4, i:4, j:4);
      WRITE (d,'    a[i]:  ');
      FOR k:=1 TO N DO BEGIN
        IF (k=i) OR (k=j) THEN BEGIN
          WRITE (d,CHR(27),'-',CHR(1));  (* Unterstreichen an *)
          WRITE (d,a[k]:4);
          WRITE (d,CHR(27),'-',CHR(0))   (* Unterstreichen aus *)
        END (* von IF *)
        ELSE WRITE (d,a[k]:4)
      END; (* von FOR k *)
      WRITELN (d);
      WRITELN(d)
    END; (* von Ausgabe *)

  PROCEDURE Sort (l, r : INTEGER);
    VAR i, j : INTEGER;
        w : INTEGER;

    BEGIN
      FOR i:=l TO r-1 DO
        FOR j:=r DOWNTO i DO BEGIN
          IF a[i] > a[j] THEN BEGIN
            w := a[i]; a[i] := a[j]; a[j] := w;
            Ausgabe(l,r,i,j)
          END (* von IF *)
        END (* von FOR j *)
    END; (* von Sort *)

  BEGIN (* Hauptprogramm *)
    REWRITE (d, 'PRINTER:');
    WRITELN ('Geben Sie ',N:3,' Zahlen ein.');
    FOR i:=1 TO N DO READLN (a[i]);
    WRITELN (d,'Anfang:');
    WRITE (d,'                                   ');
    FOR i:=1 TO N DO WRITE (d,a[i]:4);
    WRITELN(d);
    Sort (1,N);
    WRITELN (d,'Ende:');
    WRITE (d,'                                   ');
    FOR i:=1 TO N DO WRITE (d,a[i]:4);
    WRITELN(d);
    CLOSE (d)
  END.
```

Aufgabe: Testen Sie auch dieses Programm mit der Beispielzahlenfolge aus Kap. 13.1!

Im Gegensatz zu 21 Vertauschungen in der alten Version, braucht unser Programm nun nur noch 11 Vertauschungen vorzunehmen.

Übungen:

13.2.1 Schreiben Sie ein Programm zum Sortieren von 80 Ziffern, die zufällig erzeugt werden. Die Vertauschungen sollen auf dem Bildschirm sichtbar gemacht werden.

13.2.2 Schreiben Sie ein Programm, das 100 zufällig erzeugte Zeichen vom Typ CHAR sortiert. Ermittleln Sie die mittlere Zeit für 20 Programmläufe.

13.3 Quicksort (rekursiv)

Rekursion

Als dritter Sortieralgorithmus soll nun noch ein rekursiver Algorithmus behandelt werden, der gerade bei großen Datenmengen besonders schnell ist: Quicksort.

Hierbei handelt es sich um einen rekursiv geschriebenen Algorithmus.

Algorithmus

Die (rekursive) Prozedur Sort(l,r) sortiert einen Teil des Datensatzes angefangen bei einem linken Element (l) bis zu einem rechten Element (r). Beim ersten Aufruf der Prozedur (im Hauptprogramm) sind l und r natürlich die erste und letzte Nummer.

Nun wird der Datensatz unterteilt. Dazu wird ein beliebiges, hier das mittlere Element gesucht (x:= a [(l+r) DIV 2]).

Als nächstes wird der Datensatz, von links (mit dem Index i) und von rechts (mit dem Index j) kommend, durchsucht, bis sich die Indizes i und j treffen. Dabei wird x mit einem linken Element getauscht, wenn dieses größer als x ist, und x wird mit einem rechten Element getauscht, wenn dieses kleiner als x ist.

Nach dieser Schleife sind im linken Teil nur Elemente, die kleiner oder gleich und im rechten Teil nur Elemente, die größer oder gleich sind als x.

Die Prozedur ruft sich selber zweimal auf, damit nach dem gleichen Schema die beiden Teile des Datensatzes weiter sortiert werden, solange der entsprechende Teil aus mehr als einem Element besteht.

Programm

Als Programm stellt sich der Algorithmus folgendermaßen dar:

```
PROGRAM Quicksort;

  CONST N = 8;
  VAR  i : INTEGER;
       a : ARRAY [1..N] OF INTEGER;

  PROCEDURE Sort (l, r : INTEGER);
    VAR i, j : INTEGER;
        x, Hilf : INTEGER;

    BEGIN
      i := l; j := r;
      x := a[ (l+r) DIV 2 ];     (* mittleres Element *)
      REPEAT
        WHILE a[i] <= x DO i := i + 1;
        WHILE x <= a[j] DO j := j - 1;  (* Annaehern *)
        IF i < j THEN BEGIN
          Hilf := a[i]; a[i] := a[j]; a[j] := Hilf; (* Tauschen *)
          i := i + 1; j := j -1
        END (* von IF *)
      UNTIL i >= j;
      IF l < j THEN Sort (l,j);
      IF i < r THEN Sort (i,r)
    END; (* von Sort *)

  BEGIN (* Hauptprogramm *)
    WRITELN ('Geben Sie ', N:3, ' Zahlen ein: ');
    FOR i:=1 TO N DO READLN (a[i]);
    Sort(1,N);
    WRITELN ('Sortiert: ');
    FOR i:=1 TO N DO WRITELN (a[i]:3)
  END.
```

Quicksort mit Druckausgabe

Analyse

Auch bei diesem Algorithmus wollen wir zum besseren Verständnis schrittweise ausdrucken lassen, welche Vertauschungen gerade vorgenommen werden. Außerdem ist es hier nötig, zu verfolgen, welche Sortierprozedur aufgerufen worden ist. Dazu wieder ein Programm, das ein Protokoll des Sortierverfahrens ausdruckt:

Programm (für Drucker)

```
PROGRAM Quicksort;

  CONST N=8;
  VAR i : INTEGER;
      a : ARRAY [1..N] OF INTEGER;
      d : INTERACTIVE;  (* Drucker *)
      u : BOOLEAN;

  PROCEDURE Ausgabe (l,r,i,j,x : INTEGER);
    VAR k : INTEGER;
    BEGIN
      WRITE (d,CHR(27),'-',CHR(1));          (* Unterstreichen an *)
      WRITE (d,'   l:  r:  i:  j:  x:          ');
      FOR k:=1 TO N DO WRITE (d,k:4);
      WRITE (d,CHR(27),'-',CHR(0));          (* Unterstreichen aus *)
      WRITELN (d);
      WRITE (d,l:4, r:4, i:4, j:4, x:4);
      WRITE (d,'    a[i]:  ');
      FOR k:=1 TO N DO BEGIN
        IF u THEN IF (k=i) OR (k=j) THEN WRITE (d,CHR(27),'-',CHR(1));
        WRITE (d,a[k]:4);
        WRITE (d,CHR(27),'-',CHR(0))
      END; (* von FOR k *)
      WRITELN (d)
    END; (* von Ausgabe *)

  PROCEDURE Sort (l, r : INTEGER);
    VAR i, j : INTEGER;
        x, w : INTEGER;

    BEGIN
      WRITELN (d,'Sort (',l:2,',',r:2,')');
      i := l; j := r;
      x := a[(l+r) DIV 2];
      REPEAT
        u := FALSE;
        WHILE a[i] <= x DO BEGIN Ausgabe(l,r,i,j,x); i := i+1 END;
        WHILE x <= a[j] DO BEGIN Ausgabe(l,r,i,j,x); j := j-1 END;
        IF i < j THEN BEGIN
          w := a[i]; a[i] := a[j]; a[j] := w;
          u := TRUE;
          Ausgabe(l,r,i,j,x);
          i := i+1; j := j-1
        END (* von IF *)
      UNTIL i >= j;
      IF l < j THEN Sort (l,j);
      IF i < r THEN Sort (i,r)
    END; (* von Sort *)

  BEGIN (* Hauptprogramm *)
    REWRITE (d, 'PRINTER:');
    WRITELN ('Geben Sie ',N:3,' Zahlen ein.');
    FOR i:=1 TO N DO READLN (a[i]);
    WRITELN (d,'Anfang:');
    WRITE (d,'                                    ');
    FOR i:=1 TO N DO WRITE (d,a[i]:4);
    WRITELN(d);
    Sort (1,N);
    WRITELN (d,'Ende:');
    WRITE (d,'                                    ');
    FOR i:=1 TO N DO WRITE (d,a[i]:4);
    WRITELN(d);
    CLOSE (d)
  END.
```

Aufgabe: Testen Sie auch dieses Programm mit der Beispielzahlenfolge aus Kap. 13.1!

Bewertung

Der Quicksortalgorithmus nimmt bei unserer Testzahlenfolge (gleiche Zahlenfolge für alle drei Programme) nur 9 Vertauschungen vor. Es scheint also, als sei Quicksort der schnellste Algorithmus.

Ergebnis

So einfach können wir uns diese Bewertung jedoch nicht machen. Zur Untersuchung der Qualität eines Sortieralgorithmus müssen wir mit unterschiedlich großen und unterschiedlich geordneten Datenmengen arbeiten, sowie unterschiedliche Datentypen untersuchen. Außerdem gibt es noch einige andere Sortieralgorithmen, die hier aus Platzgründen nicht aufgeführt werden können.

Eine genauere Analyse der bekannten Sortieralgorithmen gibt N. Wirth in seinem Buch „Algorithmen und Datenstrukturen".

Bewertung der Algorithmen

Für unseren täglichen Bedarf an Sortieralgorithmen können wir festhalten, daß sich Quicksort oft als der schnellste Algorithmus bei größeren Datenmengen herausstellt. Bubblesort dagegen ist wegen seiner Einfachheit zu empfehlen, wenn es nicht so sehr auf die Rechenzeit ankommt. Immerhin ist Bubblesort einer der schnelleren Algorithmen.

Übungen:

13.3.1 Schreiben Sie ein Programm zum Sortieren von 80 Zahlen, die zufällig erzeugt werden. Die Vertauschungen sollen auf dem Bildschirm sichtbar gemacht werden.

Stellen Sie die rekursiven Aufrufe der Prozedur durch weitere Bildschirmzeilen (Teildatensätze) dar.

13.3.2 Schreiben Sie ein Programm, das 100 zufällig erzeugte Zeichen vom Typ CHAR sortiert. Ermitteln Sie die mittlere Zeit für 20 Programmläufe.

13.4 Zusammenfassung

Wir haben drei in ihrem Aufbau und in ihrer Geschwindigkeit verschiedene Sortieralgorithmen kennengelernt. Sie sollten sich bemühen, alle drei Algorithmen zu verstehen (mindestens aber die beiden iterativen).

Weiterhin können nun diese Sortieralgorithmen in andere Programme, in denen zur Problemlösung das Sortieren dazugehört, übernommen werden. Bei der Übernahme der Sortierprogramme (z.B. als Prozeduren) ist darauf zu achten, daß die Variablentypen zusammenpassen.

Anhang

Anhang A

1. Systemstart mit einem Laufwerk

Diskette APPLE3: in das Laufwerk einlegen und Rechner einschalten. Nach erfolgreichem Systemstart fordert der Rechner:

```
PUT IN APPLE0: AND PRESS RESET
```

Die Diskette herausnehmen und durch die Diskette APPLE0: ersetzen. Nun den <Reset>-Knopf (evtl. zusammen mit der Taste CTRL) drücken. Das System meldet sich mit dem Begrüßungstext und der COMMAND-Zeile.

2. Systemstart mit zwei Laufwerken

Diskette APPLE1: in Laufwerk Nr. 1 und andere Pascaldiskette (in der Regel APPLE2:) in Laufwerk Nr. 2 einlegen. Den Rechner einschalten. Nach erfolgreichem Systemstart meldet sich der Rechner mit dem Begrüßungstext und der COMMAND-Zeile.

3. Formatieren von Disketten

Neue Disketten müssen formatiert werden, bevor sie benutzt werden können. Dazu müssen Sie den Systemstart durchführen (sofern nicht schon geschehen).

Die Diskette APPLE3: in eines der Laufwerke legen.

Aus der COMMAND-Zeile X für eX(ecute wählen.

Wenn APPLE3: in Laufwerk Nr. 1 ist,
dann mit '#4:FORMATTER' antworten.

Wenn APPLE3: in Laufwerk Nr. 2 ist,
dann mit '#5:FORMATTER' antworten.

Alle Antworten mit <Return> abschließen.

Nun die Diskette(n) aus dem (den) Laufwerk(en) nehmen und mit 4 oder 5 antworten, je nachdem, in welchem Laufwerk die neu zu formatierende Diskette liegt.

Zum Beenden des Formatierens nur die <Return>-Taste drücken, die Diskette APPLE1: in Laufwerk Nr. 1 einlegen und nochmals <Return> drücken.

Hinweis: Neu formatierte Disketten haben alle den Namen BLANK:.

4. Kopieren von Disketten

Wenn es nötig ist, müssen Sie den Systemstart durchführen und dann aus der COMMAND-Zeile F für F(ile wählen.

Im Filer nun T für T(ransfer wählen.

Ein Laufwerk: Namen des Originals und Namen der Kopie durch Komma getrennt eingeben und mit <Return> abschließen.

Den Anweisungen des Rechners folgen und die geforderte Diskette einlegen (mit der Leertaste abschließen).

Zwei Laufwerke: Original in Laufwerk Nr. 1 und Kopie in Laufwerk Nr. 2 einlegen. #4:,#5: eingeben und mit <Return> abschließen.

Für weitere Kopien ab der Eingabe von T wiederholen.
Rückkehr in die COMMAND-Zeile mit Q für Q(uit.

5. Groß-/ Kleinschreibung

Sofern es sich um einen normalen APPLE II handelt, erreicht man die Großbuchstaben mit Ctrl-W (für einen Buchstaben) und mit Ctrl-E (für Groß-/Kleinumschaltung). Hat man einmal eine der beiden Tastenkombinationen gedrückt, so erscheinen Großbuchstaben in inverser und Kleinbuchstaben in normaler Darstellung. Auf einem Drucker werden die Zeichen jedoch in Groß-/Kleindarstellung ausgegeben.

Bei modifizierten Tastaturen, 80-Zeichen-Karten und beim APPLE IIe erreicht man die Großbuchstaben (wie bei der Schreibmaschine) mit der <Shift>-Taste.

6. Besondere Tastenkombinationen

Ctrl-A heißt: Drücke die <CTRL>- und die <A>-Taste gleichzeitig.

Ctrl-A: Umschaltung auf andere Bildschirmhälfte bei 40-Zeichen-Darstellung (normaler APPLE)

Ctrl-Z: Das 40-Zeichen-Bildschirmfenster folgt dem Cursor.

Shift-M: Eckige Klammer auf (bei normalem APPLE).

Ctrl-K: Eckige Klammer zu (bei normalem APPLE).

Durch Modifikationen an der Tastatur oder durch 80-Zeichen-Karten, bzw. beim APPLE IIe, können einige der Zeichen direkt auf der Tastatur gefunden werden oder sind umbenannt worden.

Anhang B

Hauptkommando (COMMAND-Zeile):

Command: E(dit,R(un,F(ile,C(omp,L(ink,X(ecute,A(ssem,D(ebug, ? (1.1)

(Zweiter Teil der COMMAND-Zeile wird mit ? erreicht.)

F(ile: Der FILER wird aufgerufen, das System, mit dem die Disketteninhalte angesehen, Programme, Daten oder ganze Disketten kopiert werden können.

E(dit: Der EDITOR wird aufgerufen. Mit dem EDITOR lassen sich Programmtexte erstellen, verändern und ansehen.

C(omp: Der COMPILER wird aufgerufen und ausgeführt, d.h., es wird ein Pascalprogramm (Textfile) übersetzt und ist danach lauffähig (Codefile).

R(un: Mit diesem Befehl wird das Programm, das unter dem Namen SYSTEM.WRK.TEXT gespeichet ist, der sogenannte Workfile gestartet. Sollte er noch nicht übersetzt sein, so wird automatisch der Compiler aufgerufen und das Programm übersetzt und als SYSTEM.WRK.CODE abgespeichert.

X(ecute: Diesen Befehl benutzen wir, um ein beliebiges Programm (das natürlich als Codefile vorliegen muß) auszuführen. Der Rechner fragt nach dem Namen des Programms.

U(ser: U(ser Restart ist ein Befehl, um das zuletzt ausgeführte Programm noch einmal auszuführen.

H(alt: Hat den gleichen Effekt wie das Drücken der Reset-Taste. Das ganze Pascal-System wird (wie beim Einschalten) neu aufgebaut (gebootet).

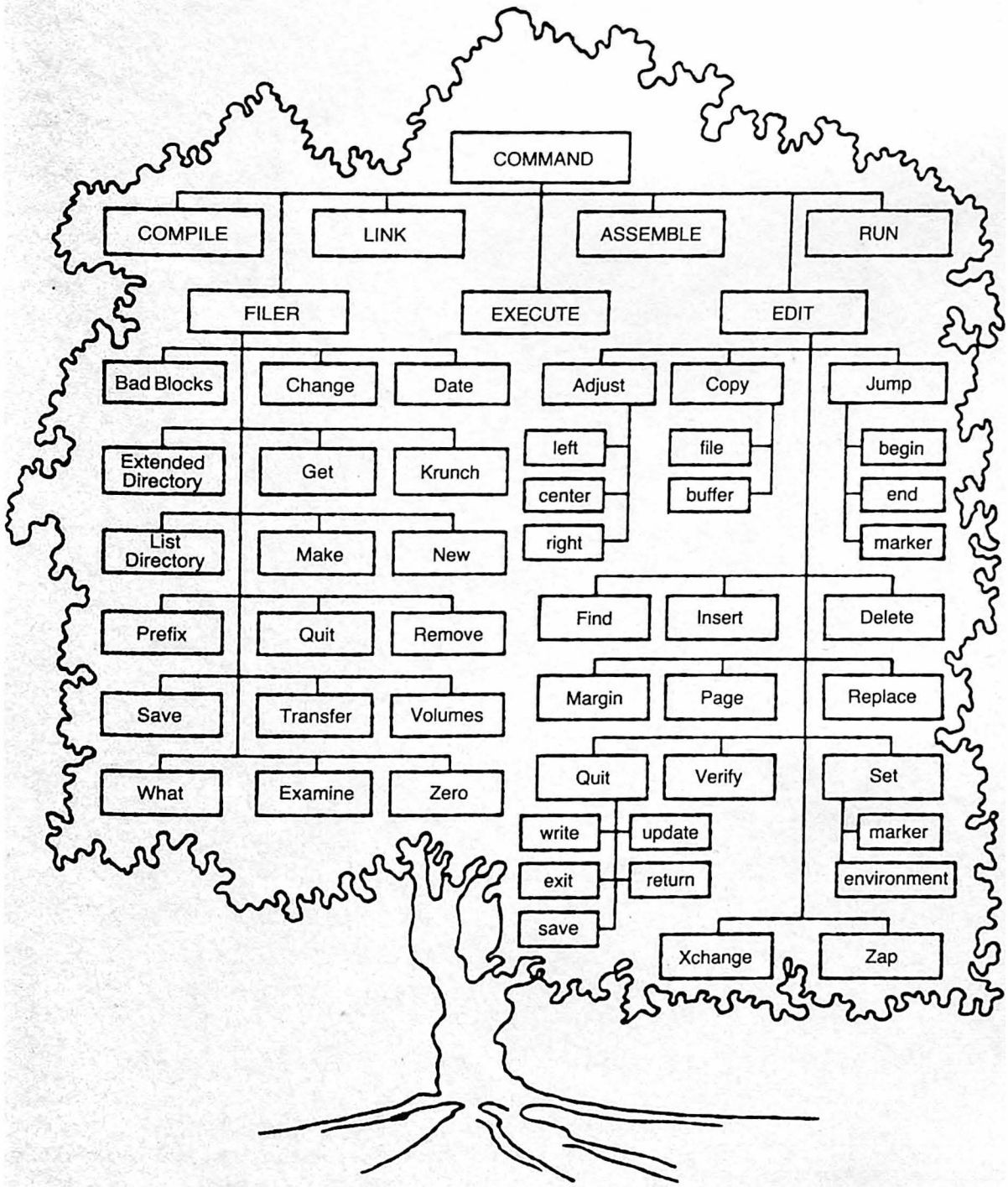
I(nit: Das Pascal-System wird initialisiert, d.h., der Rechner lernt neue Informationen über das Betriebssystem, wenn z.B. eine neue Systemdiskette eingelegt wurde oder diese verändert wurde.

A(ssem: Ruft den Pascal-ASSEMBLER auf, um Assembler-Programme zu übersetzen.

L(ink: Ruft den LINKER auf, um Assembler-Programme an Pascal-Programme anzuhängen.

D(ebug: Ruft den DEBUGGER, eine Hilfe bei der Fehlersuche, auf. Bei APPLE (noch) nicht vorhanden.

Befehlsbaum des UCSD-Betriebssystems (nach Grant, Butah: Introduction to the USCD p-System):



Editor

>Edit: A(djst C(py D(lete F(ind I(nsrt J(mp R(place Q(uit X(chng Z(ap

A(djst: Verschieben einer Zeile, z.B. zum Randausgleich, mit L linksbündig, mit R rechtsbündig, mit C mittenzentriert und beliebige Verschiebung mit den Pfeiltasten.

C(py: Kopieren eines Textes an der Cursorposition.
Mit B wird der zuletzt gelöschte Text kopiert.
Mit F kann ein ganzer Textfile eingefügt werden.

D(lete: Löschen von Zeichen oder Zeilen mit den Pfeiltasten für ein Zeichen oder mit der <Return>-Taste für eine Zeile.

F(ind: Finden von Textmustern. Das Muster wird zwischen Schrägstrichen geschrieben.

I(nsrt: Einfügen von Text an der Cursorposition.
Einrücken: Mit Leertaste. Zurück mit Pfeil-links.

J(mp: Springen im Text,
mit B an den Anfang
mit E an das Ende.

R(place: Ersetzen von Textmustern. Zu änderndes Textmuster und Änderung werden jeweils zwischen Schrägstrichen geschrieben.

Q(uit: Verlassen des Editors,
mit den Möglichkeiten:

U(PDATE THE WORK-FILE AND LEAVE	Workfile erstellen
E(XIT WITHOUT UPDATING	Nur Verlassen des Editors
R(ETURN TO THE EDITOR WITHOUT..	Nichts wird verändert
W(RITE TO A FILE NAME AND..	Speichern unter einem Namen
S(AVE WITH SAME NAME AND..	Mit gleichem Namen speichern

Die Standardwahl wird U sein.

X(chng: Auswechseln von Zeichen an der Position des Cursors.

Z(ap: Löschen eines größeren Textabschnitts (Vorsicht!)

Alle Editor-Befehle werden mit Ctrl-C abgeschlossen. Soll der Befehl doch nicht ausgeführt werden, so wird die ESC-Taste gedrückt.

Bewegen im Text: Mit Pfeil-links und Pfeil-rechts je ein Zeichen, mit P seitenweise, mit <Return> zeilenweise.

Die Bewegungsrichtung wird in der oberen linken Ecke angezeigt und kann mit < (rückwärts) und > (vorwärts) geändert werden.

Bewegung des Cursors nach oben (unten) mit Ctrl-O (Ctrl-L).

Filer

G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, Q(uit

Nach Drücken der ?-Taste erscheint der zweite Teil der Zeile:

B(ad-blks, E(xt-dir, K(rnch, M(ake, P(refix, V(ols, X(amine, Z(ero

G(et: Mit diesem Befehl wird ein File zum Workfile gemacht.

S(ave: Speichert den Workfile (SYSTEM.WRK.TEXT und SYSTEM.WRK.CODE) unter einem anzugebenden Namen ab.

N(ew: Löscht den aktuellen Workfile.

L(dir: Listet das Directory (das Inhaltsverzeichnis) einer Diskette. Wenn das Listing auf den Drucker statt auf den Bildschirm gehen soll, so hängen Sie ,#6: an den Namen;

z.B.: DISK2:,#6: oder #4:,#6:

R(em: Löschen von Files auf einer Diskette.

C(hng: Verändert den Namen einer Diskette oder eines Files auf einer Diskette.

T(rans: Mit Transfer werden einzelne Files von einem Datenträger zu einem anderen geschafft oder ganze Disketten kopiert. Wenn ein Programmtext auf den Drucker gebracht werden soll, so geht das auch mit diesem Befehl:

BLANK:TEST.TEXT,#6:

D(ate: Ein Befehl zum Einstellen des Datums, unter dem ein erstelltes Programm abgespeichert werden soll.

Q(uit: Verlassen des Filers.

B(ad-blks: Mit diesem Befehl läßt sich herausfinden, ob irgendwelche Blöcke auf der Diskette nicht in Ordnung sind.

E(xt-dir: Wie L(DIR, nur ausführlicher.

K(rnch — Schreibt auf der angegebenen Diskette alle Files so zusammen, daß keine Leerräume zwischen den Files bleiben.

V(ols: — Gibt an, welche Volumes, d.h. welche Disketten und Geräte am Rechner zur Verfügung stehen.

X(amine: — Vorsicht! Nur für Fortgeschrittene.

Z(ero: — Vorsicht! Zerstört Disketteninhalt!

Volumes

Nummer:	Name:	Bedeutung:
#1:	CONSOLE:	Bildschirm und Tastatur
#2:	SYSTERM:	dito ohne Ausgabe auf dem Bildschirm
#4:	<diskettenname>:	Diskette in Laufwerk 1
#5:	<diskettenname>:	Diskette in Laufwerk 2
#6:	PRINTER:	Drucker

Files

Folgende Arten von Files finden wir auf Disketten:

SYSTEM.name	– Systemprogramm (EDITOR, COMPILER usw.)
name.TEXT	– Textfile, d.h. ein lesbarer Programmtext
name.CODE	– Codefile, d.h. übersetztes Programm
name.DATA	– Daten (z.B. Zahlen, Texte usw.)
name.FOTO	– ein Grafik-Bild
SYSTEM.WRK.TEXT	– aktuell bearbeiteter Programmtext
SYSTEM.WRK.CODE	– aktuell übersetztes Programm

Anhang C

Reservierte Wörter in Pascal (einschließlich APPLE-Pascal):

AND	EXTERNAL	LABEL	REPEAT	WHILE
ARRAY	FILE	MOD	SEGMENT	WITH
BEGIN	FOR	NIL	SET	
CASE	FORWARD	NOT	THEN	
CONST	FUNCTION	OF	TO	
DIV	GOTO	OR	TYPE	
DO	IF	PACKED	UNIT	
DOWNTO	IMPLEMENTATION	PROCEDURE	UNTIL	
ELSE	IN	PROGRAM	USES	
END	INTERFACE	RECORD	VAR	

Vordefinierte Bezeichner

ABS	GOTOXY	ORD	SIZEOF
BLOCKREAD	HALT	OUTPUT	SQR
BLOCKWRITE	INPUT	PAGE	STR
BOOLEAN	INSERT	POS	STRING
CHAR	INTEGER	PRED	SUCC
CHR	INTERACTIVE	PUT	TEXT
CLOSE	IORESULT	PWROFTEN	TREESEARCH
CONCAT	KEYBOARD	READ	TRUE
COPY	LENGTH	READLN	TRUNC
DELETE	MARK	REAL	UNITBUSY
EOF	MAXINT	RELEASE	UNITCLEAR
EOLN	MEMAVAIL	RESET	UNITREAD
EXIT	MOVELEFT	REWRITE	UNITWAIT
FALSE	MOVERIGHT	ROUND	UNITWRITE
FILLCHAR	NEW	SCAN	WRITE
GET	ODD	SEEK	WRITELN

Anhang D

1. Die wichtigsten Standardprozeduren und -funktionen in UCSD-Pascal, die in diesem Buch beschrieben werden.

Typenabkürzungen: i : INTEGER, r : REAL, c : CHAR, s : STRING, b : BOOLEAN, a : Aufzählungstyp, / : nichts

Bezeichner	Argumenttyp	bei Funktionen Ergebnistyp	Beschreibung
ABS	r oder i	wie Argument	Betrag des Arguments
CHR	i	c	Zeichen aus ASCII mit Nummer i
CONCAT	s	s	Verbindet Strings zu neuem String
COPY	(s,i,i)	s	Kopiert Teilstring aus String
DELETE	(s,i,i)	/	Löscht Teil aus String
EXIT	Prozedurname	/	Beendet Prozedur vorzeitig
GOTOXY	(i,i)	/	Setzt Cursor an bestimmte Stelle
INSERT	(s,s,i)	/	Fügt String in anderen String ein
LENGTH	s	i	Gibt Länge des Strings an
ODD	i	b	Wahr, wenn i ungerade ist
ORD	c	i	Nummer des Zeichens c in ASCII
PAGE	(OUTPUT)	/	Löscht Bildschirm
POS	(s,s)	i	Position eines Strings in anderem
PRED	c,i,b, oder a	wie Argument	Vorgänger
PWROFTEN	r oder i	r	Zehnerpotenz
READ	i,r,c,s oder /	/	Lesen ohne Zeilenvorschub
READLN	i,r,c,s oder /	/	Lesen mit Zeilenvorschub
ROUND	r	i	Rundet zu ganzer Zahl
SQR	i oder r	wie Argument	Quadrat
STR	long-i	s	Macht aus Zahl einen String
SUCC	c,i,b oder a	wie Argument	Nachfolger
TRUNC	long-i oder r	i	Ganzzahliger Teil des Arguments
WRITE	i,r,c,s oder /	/	Schreiben ohne Zeilenvorschub
WRITELN	i,r,c,s oder /	/	Schreiben mit Zeilenvorschub

2. *Prozeduren und Funktionen aus der Bibliothek, die in diesem Buch beschrieben werden.*

Werden z.B. Anweisungen für Grafiken benötigt, so muß es in der Programmzeile nach dem Programmnamen heißen:

```
USES TURTLEGRAPHICS;
```

Bezeichner	Argumenttyp	bei Funktionen Ergebnistyp	Beschreibung
		TURTLEGRAPHICS	
FILLSCREEN	SCREENCOLOR	/	Füllt Bildschirm mit Farbe
GRAFMODE	/	/	Schaltet auf Grafik
INITTURTLE	/	/	Startbefehl der Grafik
MOVE	i	/	Bewegt Turtle weiter
MOVETO	(i,i)	/	Bewegt Turtle zu Koordinaten
PENCOLOR	SCREENCOLOR	/	Setzt Zeichenfarbe
TEXTMODE	/	/	Schaltet auf Text
TURN	i	/	Dreht Turtle weiter
TURNTO	i	/	Stellt bestimmten Winkel ein
TURTLEANG	/	i	Gibt Winkel des Turtle an
TURTLEX	/	i	Gibt X-Koordinate des Turtle an
TURTLEY	/	i	Gibt Y-Koordinate des Turtle an
VIEWPORT	(i,i,i,i)	/	Setzt Bildschirmfenster
		APPLESTUFF	
BUTTON	i	b	Wahr, wenn Druckknopf gedrückt
NOTE	(i,i)	/	Erzeugt Ton
PADDLE	i	i	Fragt Handregler ab
RANDOM	/	i	Erzeugt Zufallszahl
RANDOMIZE	/	/	Startet Zufallszahlen-generator neu
TTLOUT	(i,b)	/	Setzt Bit am Game-I/O-Port
		TRANSCEND	
ATAN	ioderr	r	Arcustangens
COS	ioderr	r	Kosinus
EXP	ioderr	r	e-Funktion
LN	ioderr	r	Natürlicher Logarithmus
LOG	ioderr	r	Dekadischer Logarithmus
SIN	ioderr	r	Sinus
SQRT	ioderr	r	Quadratwurzel

Anhang E

Der ASCII-Zeichensatz

Dezimale, oktale und hexadezimale Codewerte der ASCII-Zeichen

DEZ	OKTAL	HEX	Zeichen	DEZ	OKTAL	HEX	Zeichen	DEZ	OKTAL	HEX	Zeichen	DEZ	OKTAL	HEX	Zeichen
0	000	00	NUL	32	040	20	SP	64	100	40	@	96	140	60	`
1	001	01	SOH	33	041	21	!	65	101	41	A	97	141	61	a
2	002	02	STX	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	ETX	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	EOT	36	044	24	$	68	104	44	D	100	144	64	d
5	005	05	ENQ	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	VT	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	FF	44	054	2C	,	76	114	4C	L	108	154	6C	l
13	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	3E	N	110	156	6E	n
15	017	0F	SI	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	DLE	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	DC2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	DC3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	DC4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	NAK	53	065	35	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	ETB	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	CAN	56	070	38	8	88	130	58	X	120	170	78	x
25	031	19	EM	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	SUB	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	ESC	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	FS	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	GS	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	RS	62	076	3E	>	94	136	5E	^	126	176	7E	~
31	037	1F	US	63	077	3F	?	95	137	5F	_	127	177	7F	DEL

(Anmerkung: Der ASCII-Code verwendet nur 7 Bits eines Bytes. Das höchstwertige Bit (Bit 7) ist in dieser Tabelle auf Null gesetzt. Es kann in anderen Fällen auch den Wert 1 haben. Dann ist zum dezimalen Codewert 128, zum oktalen 200 und zum hexadezimalen 80 zu addieren.)

Anhang F

Fehlersuchmeldung des APPLE-Pascal-Compilers

(Deutsche Übersetzung von H. Elstner, Saarburg.
Mit freundlicher Genehmigung.)

1: Fehler bei der Konstruktion eines einfachen Typs
2: Der erwartete Bezeichner wurde nicht gefunden
3: Das erwartete Schlüsselwort PROGRAM wurde nicht gefunden
4: Es wurde) erwartet, aber nicht gefunden
5: Es wurde : erwartet, aber nicht gefunden
6: Falsche Reihenfolge von Sprachsymbolen (eventuell fehlt nach der vorangehenden Anweisung das Semikolon)
7: Fehler in der Parameterliste
8: Das erwartete Schlüsselwort OF wurde nicht gefunden
9: Es wurde (erwartet, aber nicht gefunden
10: Fehler in der Typ-Deklaration
11: Es wurde [erwartet, aber nicht gefunden
12: Es wurde] erwartet, aber nicht gefunden
13: Das erwartete Schlüsselwort END wurde nicht gefunden
14: Die Anweisung wurde nicht richtig abgeschlossen; eventuell fehlt nach der vorangehenden Anweisung das Semikolon
15: Es wurde der Datentyp INTEGER erwartet
16: Es wurde = erwartet, aber nicht gefunden
17: Das erwartete Schlüsselwort BEGIN wurde nicht gefunden
18: Fehler im Deklarationsteil
19: Fehler in der Komponenten-Liste
20: Es wurde . erwartet, aber nicht gefunden
21: Es wurde * erwartet, aber nicht gefunden
22: Das erwartete Schlüsselwort INTERFACE wurde nicht gefunden
23: Das erwartete Schlüsselwort IMPLEMENTATION wurde nicht gefunden
24: Das erwartete Schlüsselwort UNIT wurde nicht gefunden

50: Fehlerhafte Konstante
51: Das erwartete Zuweisungssymbol := wurde nicht gefunden
52: Das erwartete Schlüsselwort THEN wurde nicht gefunden
53: Das erwartete Schlüsselwort UNTIL wurde nicht gefunden
54: Das erwartete Schlüsselwort DO wurde nicht gefunden
55: Das erwartete Schlüsselwort TO/DOWNTO wurde nicht gefunden
56: Das erwartete Schlüsselwort IF wurde nicht gefunden
57: Das erwartete Schlüsselwort FILE wurde nicht gefunden
58: Fehlerhafter Faktor (eventuell falscher Ausdruck)
59: Fehlerhafte Variable

101: Der Bezeichner wurde bereits deklariert (eventuell kein Unterschied bei den ersten acht Zeichen)
102: Die Untergrenze überschreitet die Obergrenze
103: Der Bezeichner ist nicht vom erwarteten Datentyp

104: Der Bezeichner wurde nicht deklariert
105: Vorzeichen sind nicht zugelassen
106: Es wurde eine Ziffer/Zahl erwartet
107: Der verwendete Teilmengentyp stimmt mit der Deklaration nicht überein
108: Hier ist die Verwendung einer Datei nicht erlaubt
109: Der Datentyp darf nicht REAL sein
110: Der Datentyp des Varianten-Kennzeichens (tagfield) muß skalar oder eine Teilmenge davon sein
111: Diese Angabe ist mit dem Varianten-Kennzeichen (tagfield) nicht verträglich
112: Der Typ von Zählvariablen darf nicht REAL sein
113: Ein Indextyp muß skalar oder eine Teilmenge davon sein
114: Dieser Datentyp darf nicht REAL sein
115: Der Datentyp muß skalar oder eine Teilmenge davon sein
116: Fehlerhafter Datentyp bei Parametern einer Standard-Prozedur
117: Unbefriedigter Vorwärts-Verweis
118: Vorwärts-verweisender Typ-Bezeichner wird in einer Variablen-Deklaration verwendet
119: Bei einer vorwärts-deklarierten Prozedur ist die Wiederholung der Parameterliste nicht zulässig
120: Der Ergebnistyp einer Funktion muß skalar, eine Teilmenge davon oder ein Pointer sein
121: Die Verwendung eines Datei-Werte-Parameters ist nicht erlaubt
122: Bei einer vorwärts-deklarierten Funktion ist die Wiederholung eines Ergebnistyps nicht zulässig
123: In der Funktions-Deklaration fehlt der Ergebnistyp
124: Das Fließpunktformat ist nur beim Datentyp REAL zulässig
125: Falscher Typ bei einem Parameter einer Standard-Prozedur
126: Die Anzahl der Paramerter stimmt mit der Deklaration nicht überein
127: Unzulässige Parameter-Zuweisung
128: Der Ergebnistyp der Funktion stimmt mit der Deklaration nicht überein
129: Die Datentypen der Operanden passen nicht zusammen
130: Dieser Ausdruck ist nicht vom Datentyp SET
131: Hier ist nur ein Teil auf Gleichheit zugelassen
132: Die Einbeziehung anderer Dateien (strict inclusion) ist hier nicht erlaubt
133: Der Vergleich von Dateien ist nicht zulässig
134: Unzulässiger Typ bei (einem) Operanden
135: Der Datentyp des/der Operanden muß BOOLEAN sein
136: Der Datentyp von Mengen-(SET)-Elementen muß skalar oder eine Teilmenge davon sein
137: Die Datentypen von Mengen-Elementen passen nicht zusammen
138: Der Typ der Variablen ist kein ARRAY
139: Der Datentyp stimmt mit der Deklaration nicht überein
140: Die Variable ist nicht vom Typ RECORD
141: Der Typ der Variablen muß eine Datei oder ein Pointer sein
142: Unzulässige Parameter-Manipulation
143: Unzulässiger Typ für eine Schleifen-Steuer-Variable

144: Der Ausdruck hat einen unzulässigen Typ
145: Typ-Konflikt
146: Die Zuweisung von Dateien ist nicht erlaubt
147: Marken-LABEL-Typ und Auswahl-CASE-Ausdruck passen nicht zusammen
148: Die Grenzen von Teilmengen müssen skalar sein
149: Zählvariablen müssen vom Typ INTEGER sein
150: Eine Zuweisung an diese Standardfunktion ist nicht erlaubt
151: Eine Zuweisung an eine formale Funktion ist nicht erlaubt
152: In diesem RECORD ist kein derartiges Feld
153: Typ-Fehler beim Lesen
154: Der aktuelle Parameter muß eine Variable sein
155: Die Steuer-Variable darf nicht formal sein
156: Diese CASE-Marke wurde mehrmals deklariert
157: Die CASE-Anweisung umfaßt zu viele Fälle
158: Dieser RECORD hat kein derartiges Element
159: Varianten-Kennzeichen (tagfields) vom Typ REAL und STRING sind unzulässig
160: Die vorangehende Deklaration war keine Vorwärts-Deklaration
161: Wiederholte Vorwärts-Deklaration
162: Der Parameter muß eine Konstante sein
163: Diese Komponente fehlt in der Deklaration
164: Die Zuweisung von Standardprozeduren/-funktionen ist nicht erlaubt
165: Mehrmals aufgerufene Marke
166: Mehrmals deklarierte Marke
167: Nicht deklarierte Marke
168: Nicht definierte Marke
169: Fehler bei der Mengen-(SET)-Deklaration
170: Es wurde ein Werte-Parameter erwartet
171: Eine Standard-Datei wurde nochmals deklariert
172: Diese Datei wurde nicht deklariert
174: Es wurde eine Pascal-Funktion oder -Prozedur erwartet
175: Die maximale Länge des aktuellen Parameters ist kleiner als die maximale Länge des formalen Parameters

182: Die Verwendung von UNITs in UNITs (NESTED UNITs) ist hier unzulässig
183: In dieser Ebene sind externe Deklarationen nicht zulässig
184: Im Schnittstellen-(INTERFACE)-Bereich sind externe Deklarationen nicht erlaubt
185: In UNITs sind Segment-Deklarationen unzulässig
186: Im Schnittstellen-(INTERFACE)-Bereich sind Marken (LABELs) nicht zugelassen
187: Vergeblicher Versuch, die LIBRARY zu öffnen
188: Diese UNIT wurde bei der vorangehenden USES-Deklaration nicht angegeben
189: An dieser Stelle ist die Verwendung von USES nicht zugelassen
190: Diese UNIT ist in der LIBRARY nicht enthalten
191: Die Verwendung von Privat-Dateien ist unzulässig
192: Im Schnittstellen-(INTERFACE)-Bereich fehlt USES
193: Für diese Operation ist nicht genügend Platz vorhanden
194: Diese Angabe muß zu Beginn des Programms erscheinen

195: Diese UNIT kann in das Programm nicht eingebunden werden

201: Fehler bei einer REAL-Zahl: Es wurde eine Ziffer erwartet
202: STRING-Konstanten dürfen eine Zeile nicht überschreiten
203: Die INTEGER-Konstante liegt außerhalb der zulässigen Grenzen

250: Zu viele Bezeichner
251: Zu viele Funktionen oder Prozeduren
252: Zu viele Vorwärts-Verweise
253: Die Prozedur ist zu lang
254: Die Prozedur hat zu viele zu lange Konstanten
256: Zu viele externe Bezugnahmen/Verweise
257: Zu viele externe Funktionen oder Prozeduren
258: Zu viele lokale Dateien
259: Dieser Ausdruck ist zu kompliziert

300: Die Division durch Null ist nicht zulässig
301: Für diesen Wert ist kein (CASE-) Fall vorgesehen
302: Der Index-Ausdruck liegt außerhalb der zulässigen Grenzen
303: Der zuweisende Wert liegt außerhalb der zulässigen Grenzen
304: Ein Element des Ausdrucks liegt außerhalb der zulässigen Grenzen

350: Es wurde kein DATA-Segment zugewiesen
351: Dieses Segment wurde zweimal verwendet
352: Es wurde kein Code-Segment zugewiesen
353: Eine UNIT, die nicht in der LIBRARY enthalten ist (NON-INTRINSIC-UNIT), wurde von einer UNIT aufgerufen, die in der LIBRARY eingebettet ist (INTRINSIC-UNIT)
354: Es gibt zu viele Segmente

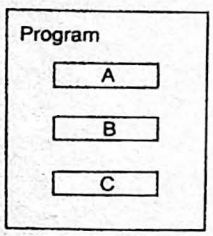
398: Implementierungs-Einschränkung
399: Implementierungs-Einschränkung
400: Der Text enthält ein unzulässiges Zeichen
401: Die Eingabe hat ein unerwartetes Ende(→ BEGIN/END i.O.?)
402: Es ist nicht genügend Platz auf der Diskette, um die Code-Datei abzuspeichern
403: Fehler beim Lesen
404: Es ist nicht genügend Platz für das Abspeichern einer Text-Datei
405: Der Aufruf in einer separaten Prozedur ist unzulässig
406: Die Angabe einer Fortsetzung-Datei (include-file) ist nicht zulässig
407: Zu viele LIBRARIES

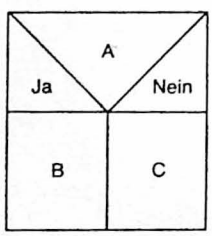
Anhang G

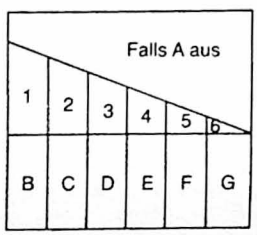
Symbole für Block- und Flußdiagramme, die verwendet werden, um bestimmte Strukturen darzustellen.

Sequenz: **Verzweigung:** **Mehrfach-Entscheidung:**

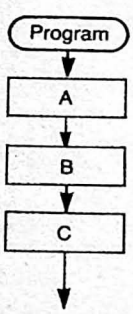
Block:

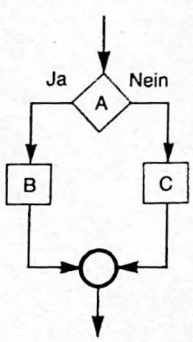


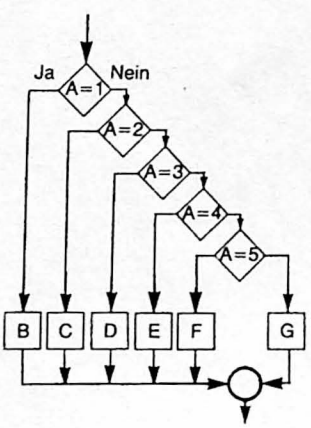




Fluß:

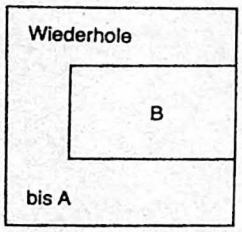


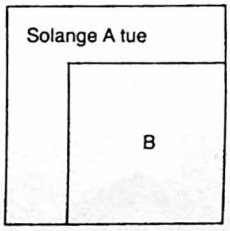




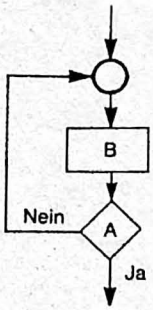
Wiederhole ... bis: **Solange ... tue ...:**

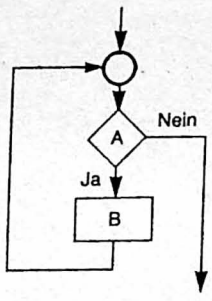
Block:





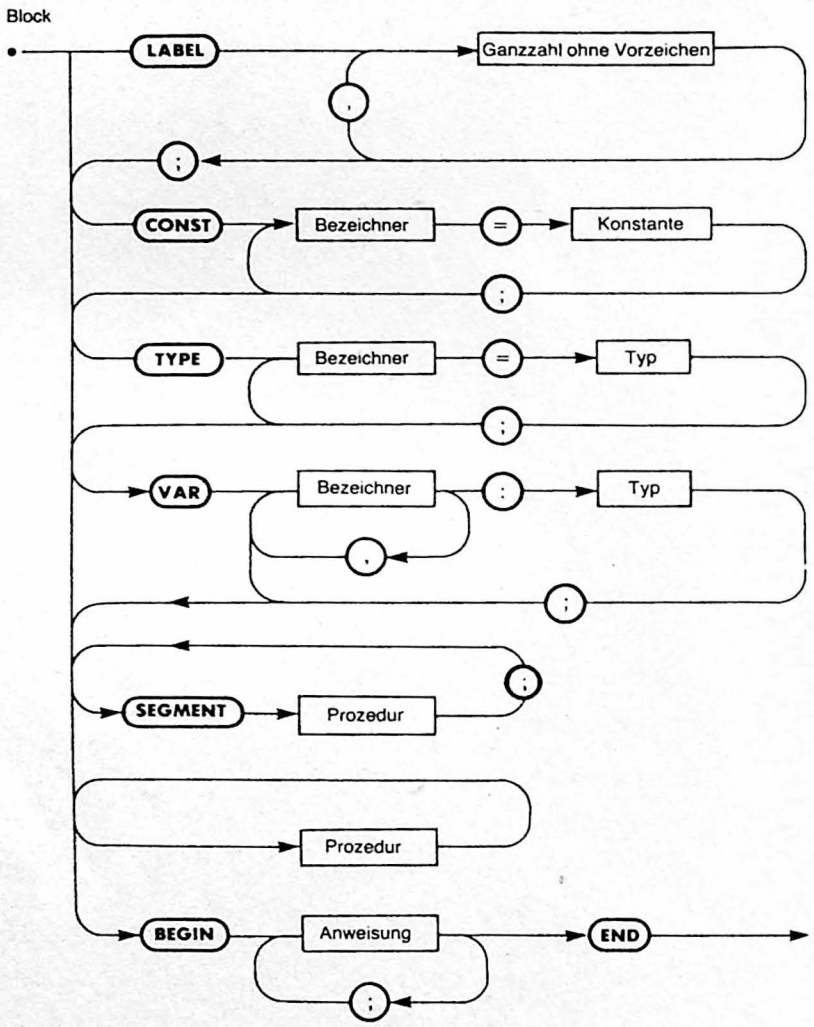
Fluß:

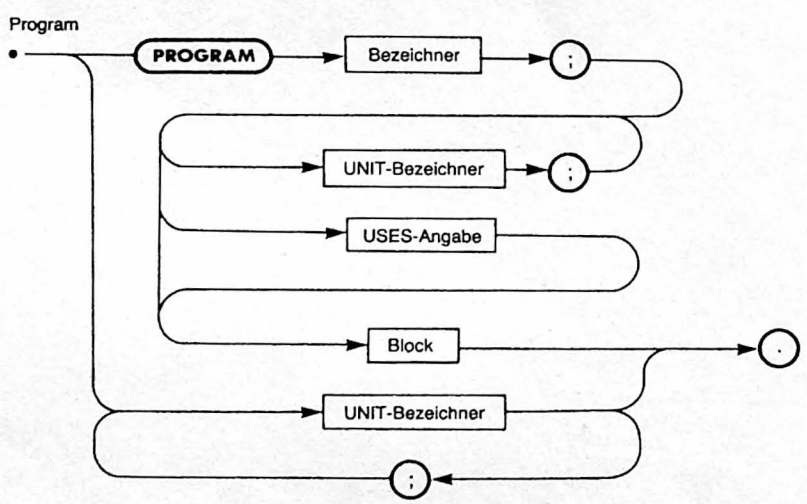




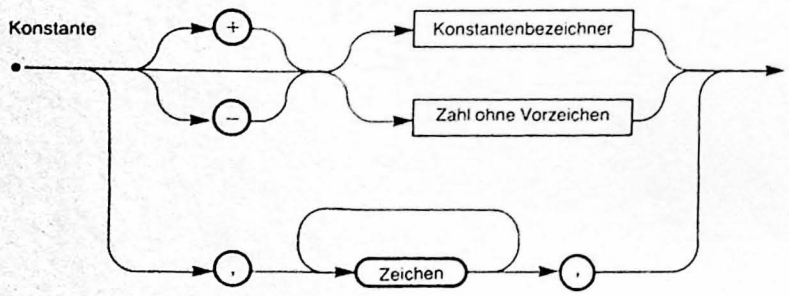
Anhang H

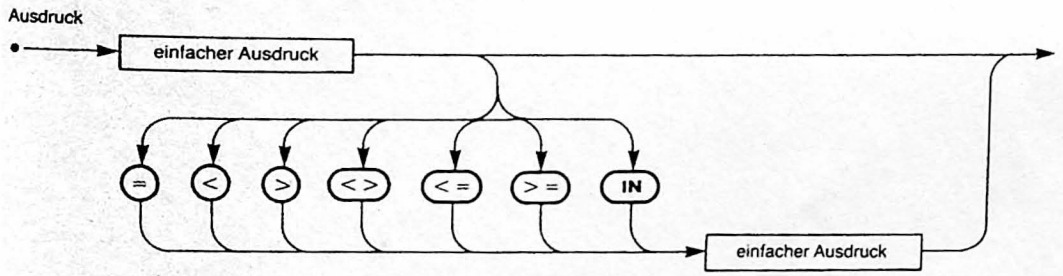
Syntaxdiagramme für UCSD-Pascal

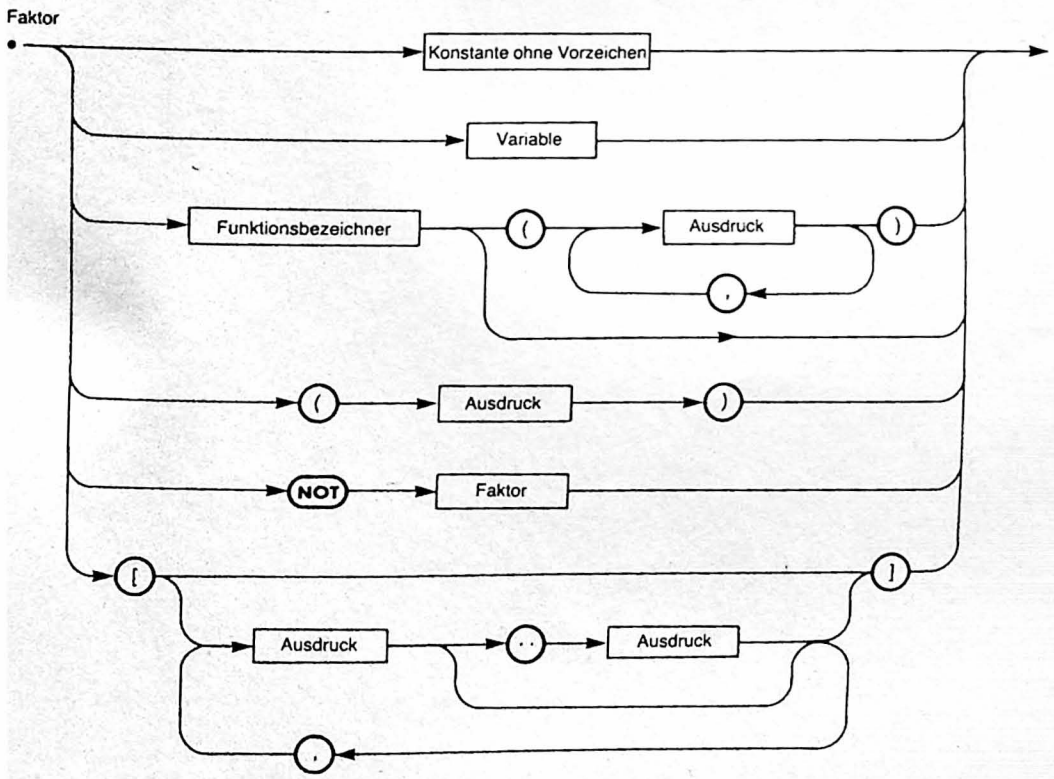




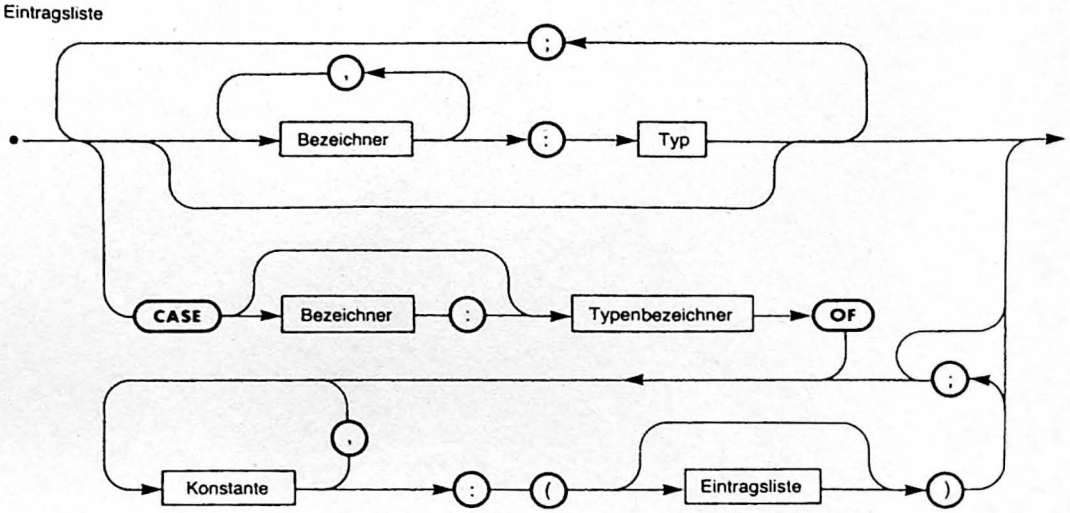
H Syntax


Konstante
+
–
Konstantenbezeichner
Zahl ohne Vorzeichen
'
Zeichen
'

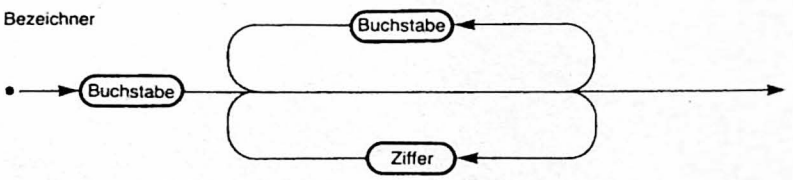

Ausdruck
einfacher Ausdruck
=
<
>
<>
<=
>=
IN
einfacher Ausdruck


Faktor
Konstante ohne Vorzeichen
Variable
Funktionsbezeichner
(
Ausdruck
)
,
(
Ausdruck
)
NOT
Faktor
[
]
Ausdruck
..
Ausdruck
,

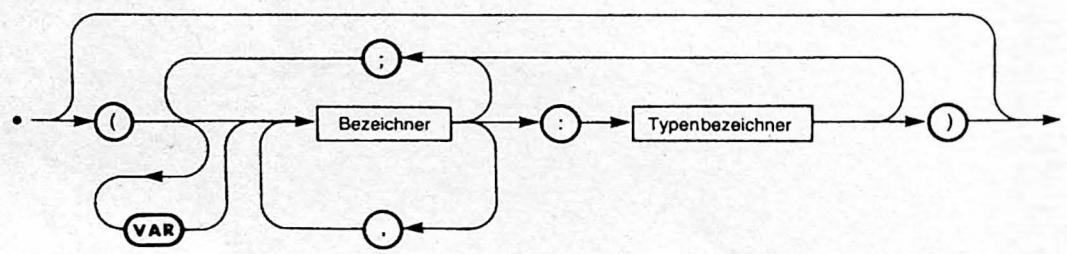
Eintragsliste



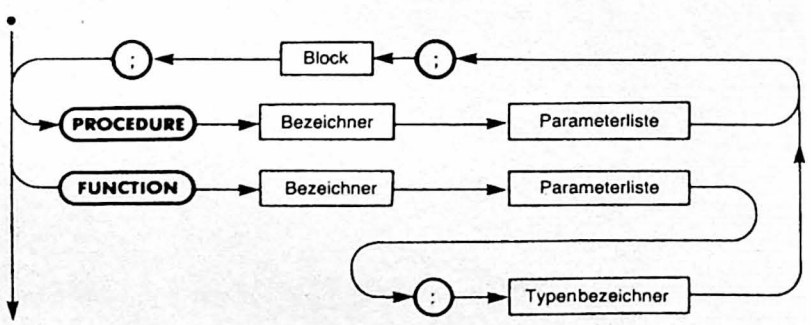
Bezeichner



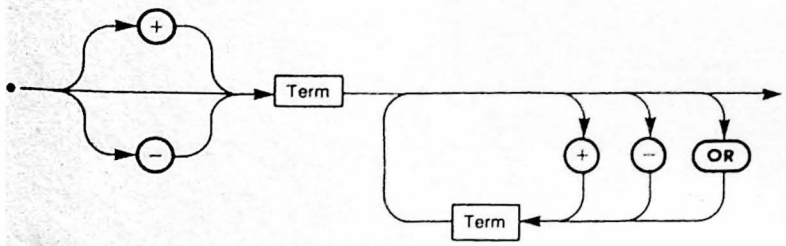
Parameterliste



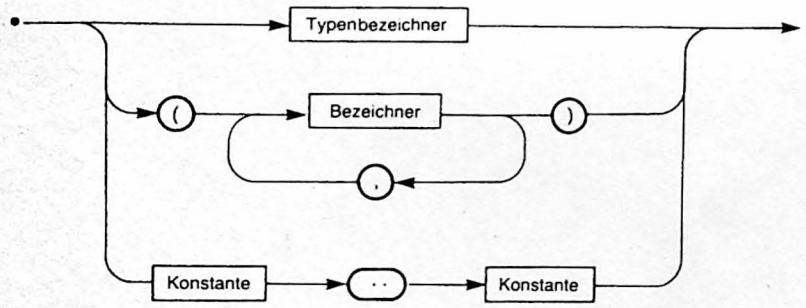
Prozedur



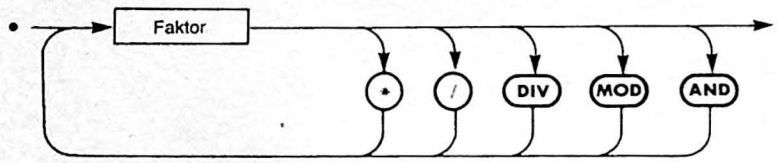
einfacher Ausdruck



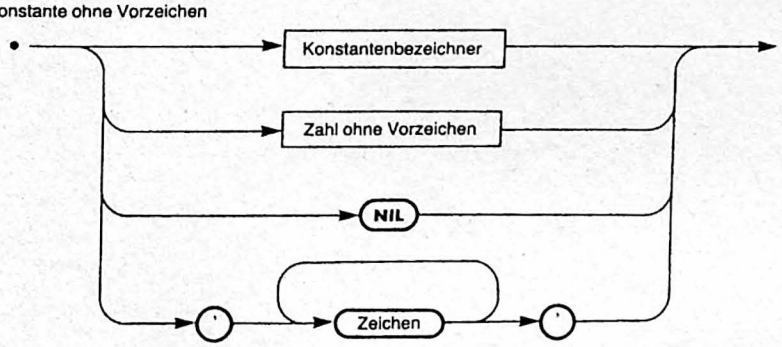
einfacher Typ



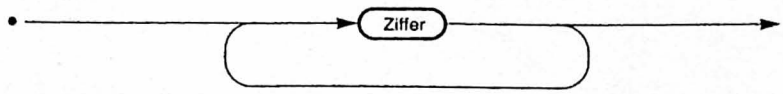
Term



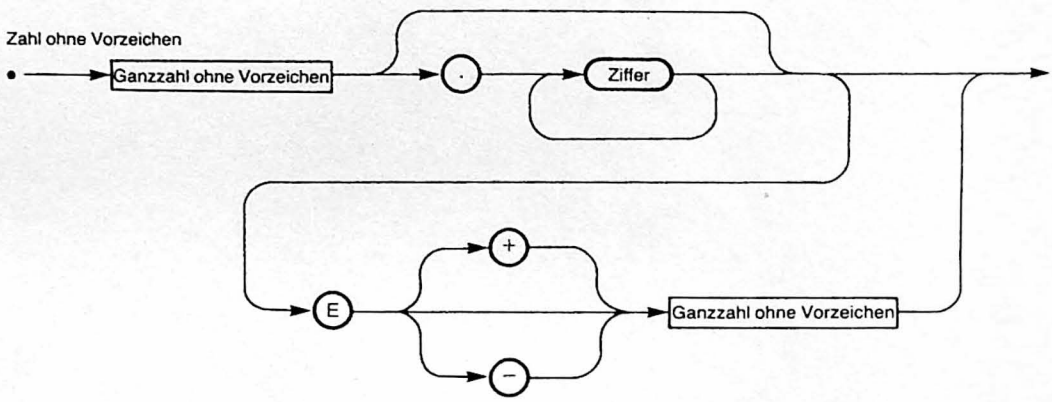
Konstante ohne Vorzeichen



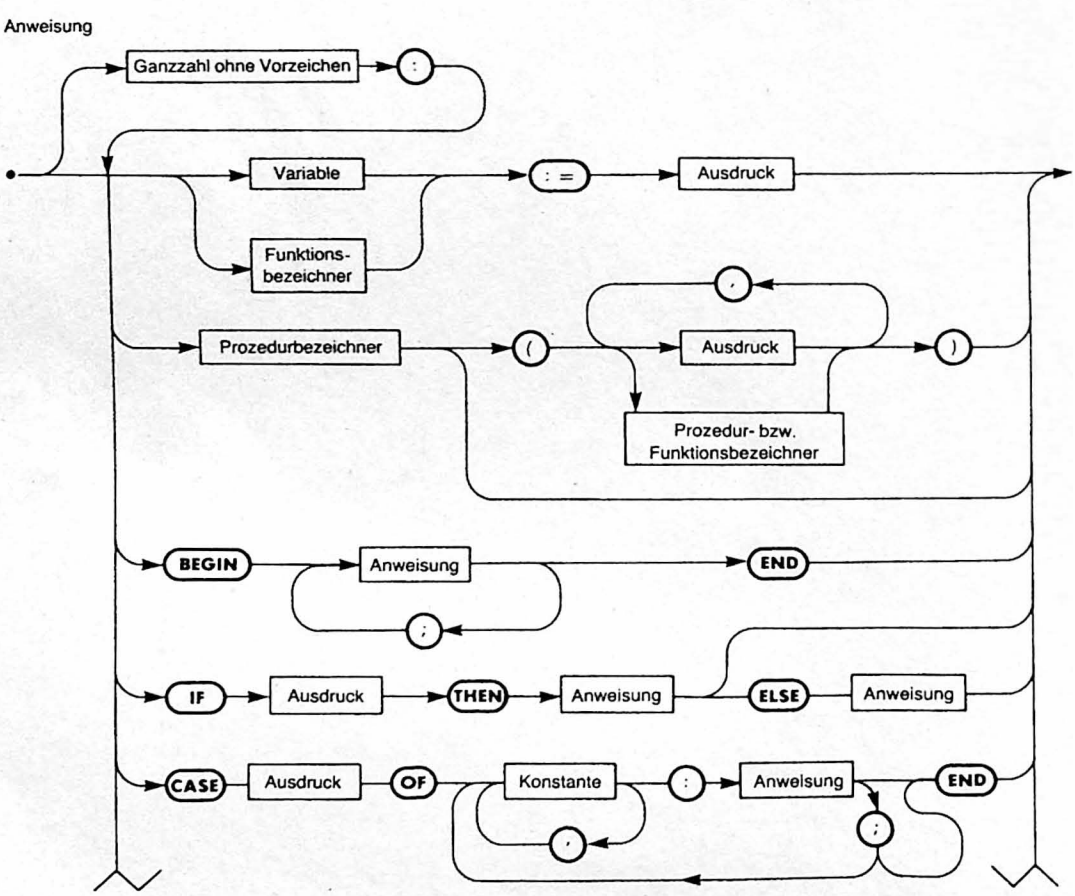
Ganzzahl ohne Vorzeichen



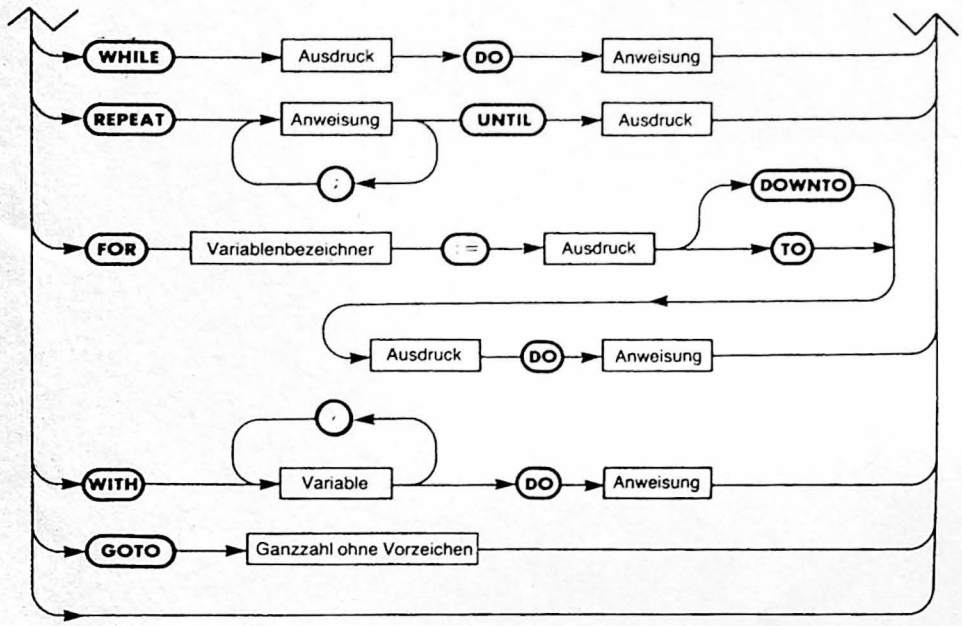
Zahl ohne Vorzeichen



Anweisung



Anweisung (Forts.)



Anhang I

Literaturverzeichnis

Baumann, R.:	Informatik mit Pascal. Klett-Verlag, Stuttgart 1981
Baumann, R.:	Programmieren in Pascal. Vogel-Verlag, Würzburg 1980
Bowles, K.L.:	Problem Solving Using Pascal. Springer-Verlag, New York 1977
Grant, Butah:	Introduction to the UCSD p-System. SYBEX-Verlag, Berkeley 1982
Grogono, P.:	Programming in Pascal. Verlag Addison-Wesley, Reading 1980
Gruber, Gutschmidt:	UCSD-Pascal 1. Interface Age, München 1981
Hergert, D., Hergert, R.:	Doing Business with Pascal. SYBEX-Verlag, Berkeley 1983
Hergert, D., Kalash, J.T.:	Apple Pascal Games. SYBEX-Verlag, Berkeley 1981
Hosseus, Sprengler, Gruner:	Pascal in Beispielen. Oldenbourg-Verlag, München, Wien 1980
Jensen, K. u. Wirth, N.:	Pascal User Manual and Report. Springer-Verlag, New York, Berlin, Heidelberg 1978
Klingen, L. H.:	Elementare Algorithmen. Herder-Verlag Freiburg, Basel, Wien 1981
Miller, A. R.:	Pascal Programme für Wissenschaftler und Ingenieure. SYBEX-Verlag, Düsseldorf 1982
Schauer, H.:	Pascal-Übungen. Oldenbourg-Verlag, München, Wien 1978
Tiberghien, J.:	Das Pascal Handbuch. SYBEX-Verlag, Düsseldorf 1983
Wilson, Addyman:	Pascal. Hanser-Verlag, München 1981
Wirth, N.:	Algorithmen und Datenstrukturen. Teubner-Verlag, Stuttgart 1979
Wirth, N.:	Systematisches Programmieren. Teubner-Verlag, Stuttgart 1979
Zaks, R.:	Einführung in Pascal. SYBEX-Verlag, Düsseldorf 1982

Anhang J

Stichwortverzeichnis

Abbruchbedingungen 93, 100, 162
ABS 46, 66, 72
Algorithmus 17, 169, 183, 186, 189, 191
AND 83
Anweisung 61
Anweisungsteil einer Funktion 151
APPLESTUFF 126
ARRAY 175, 176, 178
Ausgabe, formatierte 38, 67, 71, 73, 77, 81
Aussage 81
Aussagenlogik 81
ASCII 74
ATAN 47, 72

Bedingung 94, 99, 104
BEGIN 34
Betrag 46, 66, 72
Bezeichner 55
Bibliothek 116
Bildschirm 116, 119
Block 32
Blockdiagramm 18
BOOLEAN 41, 81
Bubblesort 188
Buchstabe 59
BUTTON 126

call-by-reference 144
call-by-value 141, 153
call-by-variable 144, 153
CASE 109
CHAR 41, 74
CHR 75
CLOSE 43
Command-Zeile 22
Compiler 16
CONCAT 79
CONST 40, 135
COPY 79
COS 47, 72
CPU 15
Ctrl-C 27, 29
Cursor 29

Datentyp 41
Deklarationsteil 32
Delete (Editor) 29
DELETE (String) 79
Dezimalzahl 69
Diskette 22
DIV 46, 65
Division 46
Doppelpunkt 23
DOWNTO 88
Drucker 30, 43

Editor 22, 26, 28
einfache Datentypen 41
Einfügen 28
END 34
Entscheidung 103, 104
Entscheidung, mehrfach 109
Ergebnis einer Funktion 150
etx 27
eXecute 22
EXIT 134
EXP 47, 72

Fallunterscheidung 110
FALSE 41, 81
Farben 119
Feld, eindimensionales 175, 176, 178
File 22, 23
Filer 22
FILLSCREEN 120
Flußdiagramm 18, 94
FOR-Schleife 88
Formatieren 24
formatierte Ausgabe 38, 67, 71, 73, 77, 81
FUNCTION 149, 151, 153
Funktion 149, 151, 153
Funktion, rekursive 163

ganze Zahl 64
Gleichheit 45
Global 40, 133, 136
GOTO 34
GOTOXY 37
Grafik 51, 115, 116, 117
GRAFMODE 118
Grundrechenarten 46

Handregler 126
Hauptprogramm 34
Hochkomma 75

IF..THEN 104
IF..THEN..ELSE 104
INITTURTLE 118
Insert (Editor) 26, 28
INSERT (String) 79
INTEGER 41, 64
INTERACTIVE 43

Jump 29

Kellerspeicher 167
KEYPRESS 126
Kommentar 33, 42
Konflikte bei Variablennamen 136
Konstante 39
Kopieren 25

Laufvariable 88, 136
LENGTH 79
LIBRARY 116
LIFO-Stack 167
Listing 43
LN 47, 72
Löschen 29
LOG 72
logischer Ausdruck 83, 104
Lokal 40, 136
Long Integer 67

Maschinensprache 16
MAXINT 64
Mehrfachentscheidung 109
Menü 20, 111
MOD 46, 65
MOVE 120
MOVETO 120

Nachfolger 64, 76
NOT 83
NOTE 127

ODD 84
OF 110
OR 83
ORD 75
Ordnung des Zeichensatzes 76

PADDLE 126
PAGE 37
Parameter einer Funktion 151
Parameter einer Prozedur 146
Parameterliste 143, 153
PENCOLOR 119
POS 79
PRED 76
PROCEDURE 130, 135, 140, 144
Programmkopf 33
Programmlisting 43
Prozedur 33, 129, 130, 135, 140, 144
Prozedur, rekursive 160, 191
Prozeduraufruf 133, 140, 145
Punkt 34
PWROFTEN 72

Quadrat 47
Quicksort 191

RAM 15
RANDOM 126, 179
RANDOMIZE 126
READ 36
READLN 36
REAL 41, 69
Rechengenauigkeit 70
Rechenzeit bei Rekursionen 165
Rekursion 159, 160, 164, 168, 191
Rekursionsabbruch 162
Rekursionstiefe 168
REPEAT-Schleife 93
Replace 29
Reservierte Worte 56
Rest 65
REWRITE 43
ROM 15
ROUND 47, 66, 72
Run 22
Rundungsfehler 70, 96

Schachtelung 91
Schachtelung von Entscheidungen 108
Schachtelung von Prozeduren 134
Schachtelung von Schleifen 91, 95
Schleife 87
Schrittweite 88
SCREENBIT 121
SCREENCOLOR 119
Semikolon 34, 95
SIN 47, 72
Solange..tue 98
Sortieren 183, 184, 188, 191
Sortieren durch Austausch 184, 188
Sortieren, rekursiv 191
Speicherplatz 39
SQR 47, 66, 72
SQRT 72
Stack 167
Stapelspeicher 167
Start 22
STR 79
STRING 41, 77
Struktur 32
SUCC 76
Symbole 57
Syntax 58
Syntaxdiagramme 59

Terminiertheit 97, 100
Text 36
TEXTMODE 118
TO 88
Tonerzeugung 127
TRANSCEND 127
Transfer 30
TRUE 41, 81

TRUNC 46, 66, 72
TTLOUT 127
TURN 120
TURNTO 120
Turtle 116
TURTLEANG 121
TURTLEGRAPHICS 116, 117
TURTLEX 120
TURTLEY 121
Typkonflikt 50, 72

UCSD-Pascal 11
UNIT 33, 116, 117, 125, 136
Unterprogramm 129
Unterschied Funktion-Prozedur 150
Unterschied REPEAT-WHILE 101
Unterschied STRING-CHAR 78
Update 27
USES 33, 116, 118, 125, 136

VAR 40, 135, 144
Variable 39, 18
Variable, indizierte 176
Variablendeklaration 40, 61, 135
Variablenname 40
Variablenübergabe bei Prozeduren 140, 144
Vergleichsoperator 65
Verzögerungsschleife 90
VIEWPORT 119
Volume 23
Vorgänger 64, 76

Wahrheitstabelle 84
Wahrheitswert 81
Wertzuweisung 45
WHILE-Schleife 98
Wiederhole...bis 93
Wiederholung 87
Workfile 24
Worte, reservierte 56
WRITE 36
WRITELN 36
Wurzel 47

Zählschleife 88
Zählvariable 89, 136
Zehnerpotenz 72
Zeichen 74
Zeichenkette 77
Ziffer 59
Zufallszahl 106, 126
Zufallszahlen 178
Zuweisung 45

SYBEX-VERLAG GMBH

4000 DÜSSELDORF 30
Postfach 300961 · Telefon 0211/61802-0 · Telex 8 588 163

SYBEX informatik

Der sichere Einstieg in PASCAL

- für den Informatik-Unterricht in der Sekundarstufe II,
- für Informatik-Kurse an der Volkshochschule,
- für Kurse und als Sekundärliteratur an der Hochschule,
- für Hobby-Programmierer und zum Selbststudium.

Dieses Buch wurde von einem Lehrer aus der Praxis geschrieben – Lehrer haben es erprobt und mitgestaltet.

In einer leicht verständlichen Sprache werden Sie mit einer vielgenutzten Programmiersprache vertraut gemacht. Sie machen Ihre ersten Schritte im UCSD-Pascal-System, lernen den Aufbau eines Pascal-Programms, reservierte Wörter, Syntaxdiagramme, Prozeduren, Funktionen, Rekursionen und Sortieralgorithmen kennen.

Den Weg durch das Buch bestimmen Sie selbst, da die Themen weitgehend unabhängig voneinander behandelt werden. Ihre neuen Kenntnisse können Sie direkt an vielen Übungen und Beispielprogrammen erproben, die auf einem Apple-Rechner getestet wurden und leicht auf andere Systeme zu übertragen sind.

Über den Autor

Karl-Hermann Rollke unterrichtet an einem Gymnasium Informatik, Mathematik und Physik. Seit mehreren Jahren arbeitet er mit Mikrocomputern und hat in der Sekundarstufe II Informatik-Kurse gehalten. Seine dabei gewonnenen Erfahrungen mit Pascal konnte er in Pascal-Kurse an der VHS und zahlreiche Veröffentlichungen in verschiedenen Fachzeitschriften einbringen.

DM 24,80
sFr 23,–
öS 193,–

9 783887 450465

Grundkurs in PASCAL Band 1

ISBN 3-88745-046-9

SYBEX informatik